(2)

Technical Report 1466
November 1991

Local Area Network Distributed Realtime Clock Synchronization

D. R. Wilcox

ADMINISTRATIVE INFORMATION

This work was performed for the Office of Naval Technology during FY 91 by the Embedded Computer Systems Branch, Code 412, Naval Ocean Systems Center (NOSC), under the Computer Technology Block Program, Command and Control Department, NOSC Code 41.

Released by
L. J. Core, Head
Embedded Computer
Systems Branch

Under authority of
A. G. Justice, Head
Information Processing
and Displaying Division

ACKNOWLEDGMENTS

The author wishes to extend special thanks to the following people:

James L. Conrath, NOSC Code 412, for technical and textual review of this report.

Viet D. Tran, NOSC Code 412, for fabrication of the hardware and creation of the software for an Am2942-based periodic phase modification realtime clock synchronization demonstration for the IEEE 802.5 local area network.

LH/JA

EXECUTIVE SUMMARY

OBJECTIVE

The objective of this work was to apply the strobe realtime clock synchronization technique (introduced in a previous report) to serial busses and local area networks in general. The work also applies the technique to the IEEE Standard 802.5 and the Fiber Distributed Data Interface token ring local area network standards in particular.

RESULTS

The strobe distributed realtime clock synchronization technique, which has already been incorporated into backplane bus standards, can also be applied to token ring local area networks. The strobe is implemented as a frame protocol data unit whosc destination address field is recognized by the strobe detector at all the participating nodes. The strobe frame source and information fields support the strobe label. The strobe technique does not require modification of existing local area network standards.

RECOMMENDATIONS

Strobe detection should be performed in hardware rather than in software because hardware minimizes the variation in the time between reception of the strobe and capture of the realtime clock value.

The realtime clock synchronization support hardware, when partitioned, should be partitioned at the interface between the strobe detector, which is driven by the local area network timing, and the adjustable realtime clock, which is driven by the clock oscillator timing. The strobe signal, which interfaces the two sections, can serve as an interrupt to notify the processor of the presence of a new realtime clock sample value.

NOTE: Section 5 highlights some of the major conclusions of this report. Sections 1 through 4 provide more detail as well as additional conclusions.

CONTENTS

FIGURES

1.0 INTRODUCTION

A previous report, *Backplane Bus Distributed Realtime Clock Synchronization* [1], presented methods of synchronizing the realtime clocks associated with a set of respective digital processing modules interconnected through a backplane bus. All the methods were based on the broadcast of a strobe signal over the backplane bus that simultaneously samples all the realtime clocks being synchronized. The report described and compared various methods of implementing realtime clocks that permit their rate of advance with respect to physical time to be adjusted. It examined and made recommendations on various adjustable rate realtime clock interface standardization issues. It also gave and evaluated software algorithms for synchronizing adjustable rate realtime clocks.

This latest report now applies the strobe realtime clock synchronization technique, introduced in the previous report, to serial busses and local area networks in general, and to the IEEE 802.5 [2] and the Fiber Distributed Data Interface (hereafter called FDDI) [3] token ring local area network standards in particular. Section 2 examines miscellaneous material relating to adjustable rate realtime clocks; much of this material was introduced by the previous report and is included here in the interest of providing greater clarity. Section 3 presents adjustable rate realtime clock hardware implementation methods considered superior to those of the previous report. Section 4 presents a hardware implementation of an IEEE 802.5 strobe detector.

This report assumes familiarity with the concepts and terminology of the previous report. It is *strongly* recommended that the reader unfamiliar with the previous report study it before reading this report. It is also recommended that the reader be familiar with the IEEE 802.5 and FDDI local area network specifications.

1.1. REALTIME CLOCK SYNCHRONIZATION HARDWARE PARTITIONING

The realtime clock synchronization hardware can be partitioned into two major sections, the strobe detector section and the realtime clock section. Figure 1 shows a block diagram of the partitioning along with the various interfaces. The internal bus interconnects the

[1] D. R. Wilcox, *Backplane Bus Distributed Realtime Clock Synchronization*, NOSC TR 1400, Naval Ocean Systems Center, Dec. 1990.

[2] IEEE Standard 802.5-1989, *Token Ring Access Method and Physical Layer Specification*, Institute of Electrical and Electronic Engineers, 345 E. 47th St., New York, NY 10017, Sept. 1989.

[3] Only need two documents for the material presented in this report: ANSI X3T9.5/88-148, *FDDI Physical Layer Protocol (PHY-2) (Maintenance Revision) Working Draft Proposed American National Standard*, American National Standards Institute, May 25, 1990, and ANSI X3T9.5/88-139, *FDDI Media Access Control (MAC-2) (Maintenance Revision) Working Draft Proposed American National Standard*, American National Standards Institute, May 25, 1990.

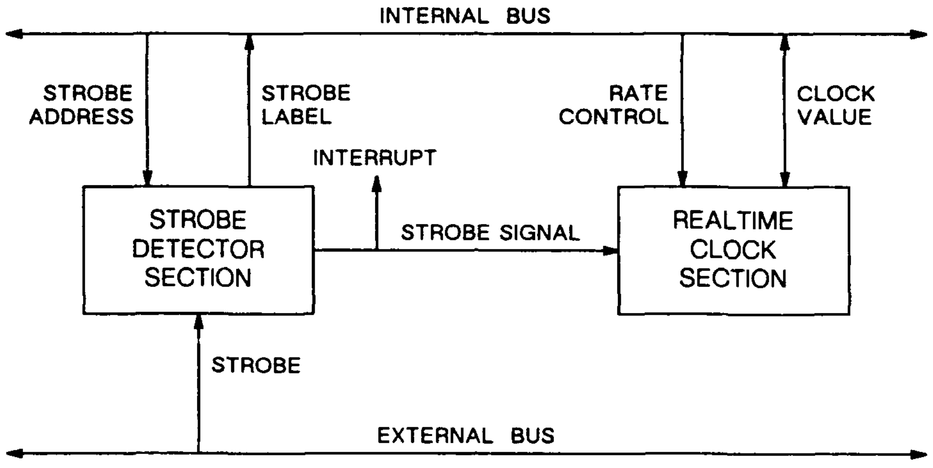


Figure 1. Realtime clock synchronization hardware partitioning.

components within the module or node. The external bus interconnects the module or node with other modules or nodes, respectively. Realtime clock synchronization strobes are broadcast over the external bus.

The strobe detector section consists of a strobe detector, an optional strobe address register, and an optional strobe label register. The strobe detector generates a strobe signal indicating when it detects broadcast to an address dedicated to the strobe appearing on the external bus. The strobe signal is sent to the realtime clock section where it captures the current time. The strobe address register makes the address dedicated to the strobe programmable by the processor. If there is no strobe address register, the recognition of a fixed strobe address is incorporated into the strobe detector. The strobe label register captures the strobe label. The strobe label uniquely identifies the strobe. The strobe label consists of two components: the identity of the module or node that generated the strobe and the strobe sequence number for the strobe from that module or node [4]. The purpose of the strobe label is to detect race conditions involving multiple strobes in complex systems [5].

The strobe detector implementation depends upon the type of external bus broadcasting the strobe. On a parallel backplane bus, the strobe detector is an address identity comparator. On a serial backplane bus or a local area network, the strobe detector is a state machine. The state machine detects the sequence of serial bits representing the strobe address. The

[4] For a backplane bus strobe label example, see IEEE P896.2, *Futurebus+ Physical Layer and Profile Specification*, Draft 5.5, July 1991, sec. 3.2.2.12.4, p. 56. (This is an unapproved document. Do not specify or claim conformance to this document.)

[5] See Wilcox, NOSC TR 1400, p. 5.

implementation of the strobe detector may also depend on the partitioning employed by the set of bus-interface integrated circuits to which it is connected.

The realtime clock section consists of the adjustable-rate realtime clock, the sample register, and one or more optional interval timers. The adjustable-rate realtime clock interfaces to the host processor through the clock counter and through registers associated with its particular method of controlling the clock rate. The sample register captures the value contained in the clock counter immediately upon receiving the strobe signal from the strobe detector.

There are various way to implement the adjustable-rate realtime clock. These include the hidden offset method [6], the periodic phase modification method [7], and the phase accumulation method [8].

The design of the two major sections are independent of one another. The only interface between them is the strobe signal. This permits the same realtime-clock-section integrated circuit design to be used with a variety of different strobe-detector-section integrated circuit designs. Conversely, it also permits the same strobe-detector-section integrated circuit design to be used with a variety of realtime-clock-section integrated circuit designs. The interconnecting strobe signal requires only a single integrated circuit pin.

The partitioning into these two major sections has an additional advantage. It isolates the circuitry driven by the clock signal derived from the serial bus or local area network from the circuitry driven by the clock signal derived from the realtime clock oscillator. These clock signals are usually asynchronous of one another. With this partitioning, the only synchronization interface point between them is through the strobe signal.

1.2. LOCAL AREA NETWORK STROBE DETECTION

Strobes are broadcast over a local area network as frame protocol data units. Figure 2 shows the frame format for both the IEEE 802.5 [9] and the FDDI [10] local area networks. The destination address field contains a broadcast address identifying the frame as a strobe. The address is assigned by the system designer, if programmable, or by the strobe detector hardware designer, if not programmable. The source address field and the information field

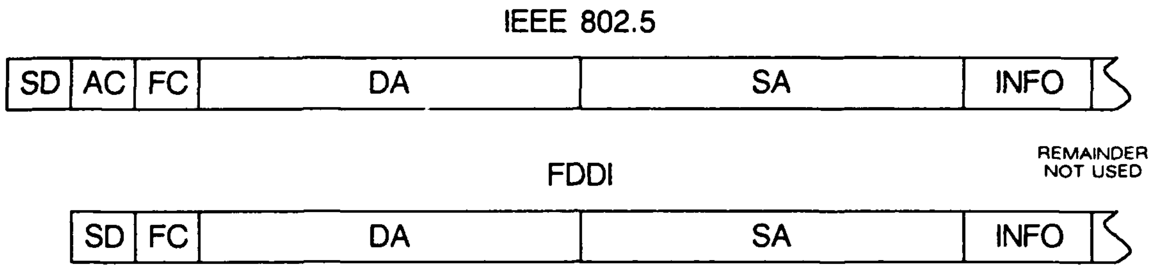
[6] See Wilcox, NOSC TR 1400, pp. 9–12.

[7] See Wilcox, NOSC TR 1400, pp. 12–19. This report will provide additional material to that already provided in NOSC TR 1400.

[8] See Wilcox, NOSC TR 1400, pp. 19–20. This method has been selected as the clock model within the IEEE P1212 Control Status Register (CSR) specification. See IEEE P1212 Draft 4.0, Sept. 22, 1990, sec. A-7.3, p. A44. (This is an unapproved draft. Do not specify or claim conformance to this document.)

[9] See IEEE 802.5-1989, sec. 3.1.2, p. 23.

[10] See ANSI X3T9.5/88-139, *FDDI Media Access Control (MAC-2)*, sec. 7.2.2, p. 33.



SD	Starting Delimiter (1 octet)
AC	Access Control (1 octet)
FC	Frame Control (1 octet)
DA	Destination Address (2 or 6 octets)
SA	Source Address (2 or 6 octets)
INFO	Information (0 or more octets)

Figure 2. Portion of frame formats used by strobe detector.

contain the strobe label. The source address field identifies the node generating the strobe. The source node local area network interface automatically generates this field when the frame is transmitted. The information field contains the strobe sequence number. While one information field octet is sufficient, support for two octets has the benefit of making the width of the strobe label an integer power of 2. The source node strobe generation software creates the strobe sequence number. The remaining fields are ignored by the strobe detector except possibly for error checking. Both IEEE 802.5 and FDDI transmit all octets, and the symbols forming the octets, most-significant bit first [11].

Since strobes are simply an application of the local area network protocol, and not a modification of the protocol, the strobe technique does not require any modification of existing local area network standards.

Reception of a strobe by a node triggers sample register capture of the local realtime clock value. Reception of a strobe must also initiate the software needed to process the recorded sample. The processor can initiate the software by simply processing the strobe frame like any other local area network application input. The software can then extract the strobe sequence number within the information field directly. Alternatively, as shown in figure 1, the processor can initiate the software through an interrupt derived from the strobe

[11] See IEEE 802.5-1989, sec. 3.2.5.3, p. 30; ANSI X3T9.5/88-139, *FDDI Media Access Control (MAC-2)*, sec. A.3, pp. 79-80.

signal emitted by the strobe detector. When the interrupt approach is used, the processor does not need to receive the local area network strobe frame itself, provided that the strobe detector section captures the entire strobe label. This means that the strobe detector section must capture the strobe sequence number contained in the information field as well as the strobe source contained in the source address field.

Local area network interface integrated circuit families generally partition their implementations at the interface between the physical layer and the media access control sublayer of the data link layer. This is done because the requirements placed upon the logic technology at the two layers are different. The physical layer hardware deals with high-speed serial data, phase lock loop clock recovery, and exotic drive requirements. The media access control sublayer hardware, on the other hand, implements highly complex, purely digital logic that can operate at a lower speed by processing local area network data in parallel.

The strobe detector hardware obtains its local area network input by monitoring the local area network receive data passing from the physical layer hardware to the media access control sublayer hardware. The strobe detector does not interrupt or modify the data passing through the interface. Its only impact is additional electrical loading on some of the local area network interface integrated circuit output pins.

Conceptually, the local area network strobe detector can be logically partitioned into two components, the octet state machine and the protocol state machine. The octet state machine uses the unique coding of the protocol data unit starting delimiter field to determine the alignment needed to convert the serial local area network receive data stream into an octet data stream. The octet data stream is sent to the protocol state machine. The protocol state machine searches for the sequence of octets defining a strobe, and when found, generates the strobe signal and captures the strobe label.

Figure 3 shows a flow chart for the protocol state machine. Each state processes one octet. States *A* through *I*, on the left side of the flow chart, detect the strobe. States *J* through *Q*, on the right side of the flow chart, capture the strobe label. State *B* is omitted for FDDI since, unlike IEEE 802.5, FDDI does not have an access control field octet in its frame format. States *F* through *I* and states *L* through *O* are omitted when 2-octet addressing is used instead of 6-octet addressing. Finally, states *P* and *Q* can be omitted when capture of the strobe sequence number within the information field is unnecessary.

The protocol state machine can be implemented using a counter. The counter output represents the state. State *A*, which waits for reception of a starting delimiter field octet, is assigned the counter reset state. The counter resets after any state fails to meet the conditions required to enter the next sequential state. For example, when the counter is in state *C*, it is looking for a valid frame control field octet. If it sees one, the counter increments to

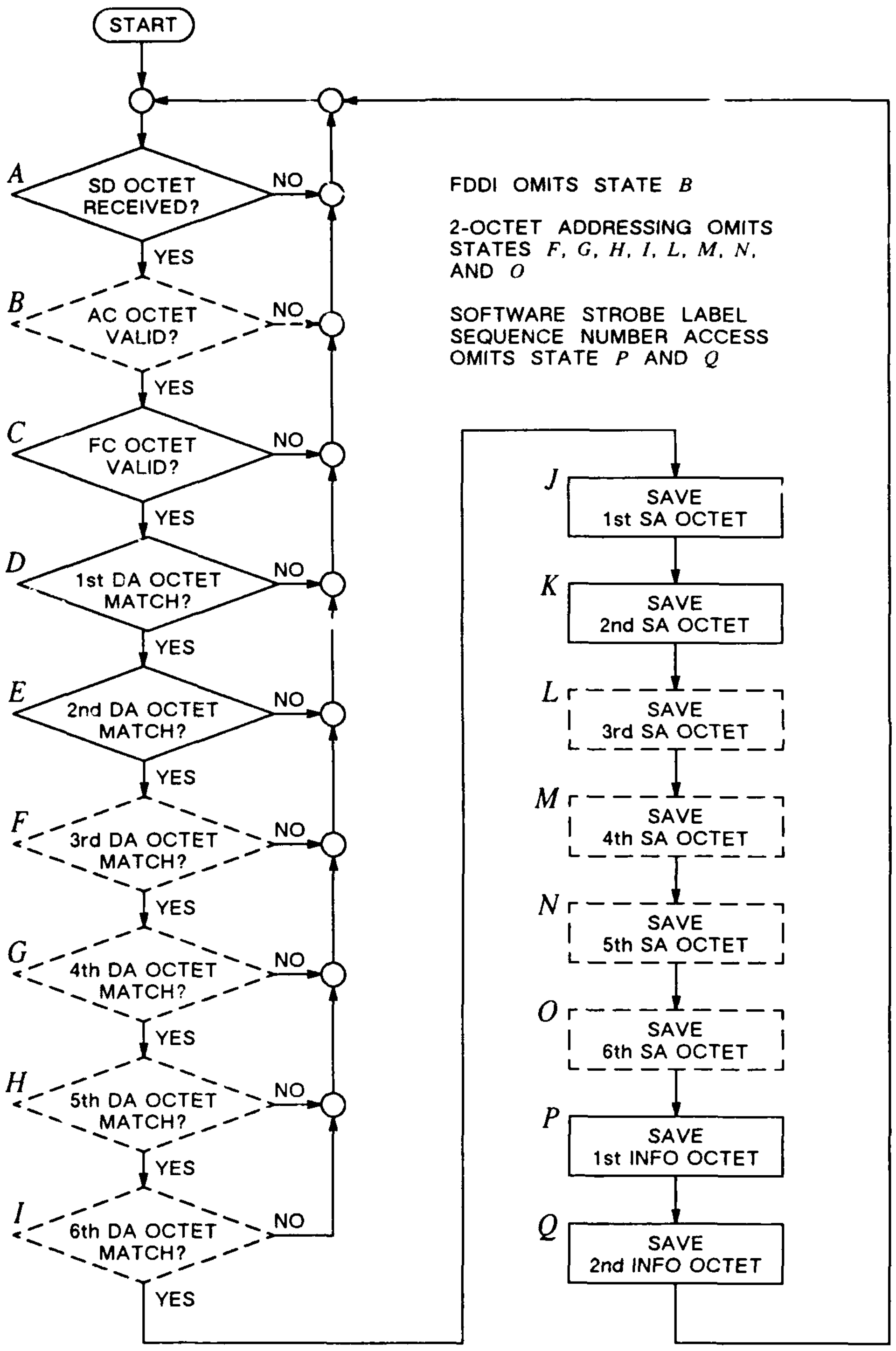


Figure 3. Protocol state machine flow chart.

state *D*; otherwise it resets to state *A*. There are no conditions placed on states *J* through *Q*. In these states, the counter always increments to the next sequential state. In the case of state *Q*, the next sequential state is state *A*.

1.3. RATIONALE FOR HARDWARE STROBE DETECTOR IMPLEMENTATION

The critical path for accurate realtime clock synchronization using the strobe technique is from the physical broadcast of the strobe on the bus to the capture of the realtime clock value in the sample register. The critical-path propagation delay includes the retransmission delay of the local area network signal through the intermediate nodes and the propagation delay through the local area network media interconnecting the nodes. It also includes delays due to the asynchronous nature of the local area network signal timing with respect to the realtime clock oscillator timing. Propagation delay through the intermediate nodes, called retransmission delay, depends upon the local area network standard and upon the integrated circuit implementation. Propagation delay through the media for fiber-optic cable is about the same as for coaxial cable, or approximately 5 microseconds per kilometer [12]. All these delays are only a problem to the extent that they are both unknown and numerically significant relative to the desired strobe sampling resolution. Local rate adjustment software can easily adjust sample register values for known delays.

Since minor delays occurring before the strobe broadcast reaches the physical bus are not in the critical path, strobes can be generated by software. They can use the same host processor local area network interface used to generate other local area network traffic. Strobe generation can be treated as an independent periodic realtime process executing under the host processor operating system. The strobe generation process does not need be incorporated within the operating system kernel nor have a high priority [13].

The strobe detector and sample register, which are in the critical path, are implemented in hardware. Software is not used because it is difficult to accurately predict the delay through the many layers of local area network, operating system, and realtime clock software interfaces. One can get a feel for the many interfacing layers involved by examining an implementation originally designed for the "statistically rambunctious Internet" [14] wide-area network environment. Figure 4 shows the critical path through the layers of the portable

[12] A typical fiber-optic cable figure of 5.085 microseconds per kilometer is given in *FDDI Physical Layer Protocol (PHY-2)*, Annex A, pp. 45–47. Note that European numeric notation uses a comma for a decimal point and a blank rather than a comma to separate three digit groups.

[13] See Wilcox, NOSC TR 1400, p. 6.

[14] See David L. Mills, *Measured Performance of the Network Time Protocol in the DARPA/NSF Internet System*, Dept. Electrical Engr., Univ. of Delaware, Newark, DE 19716, Tech. Rep. Udel-EE 89-9-3, Sept. 1989, p. 15.

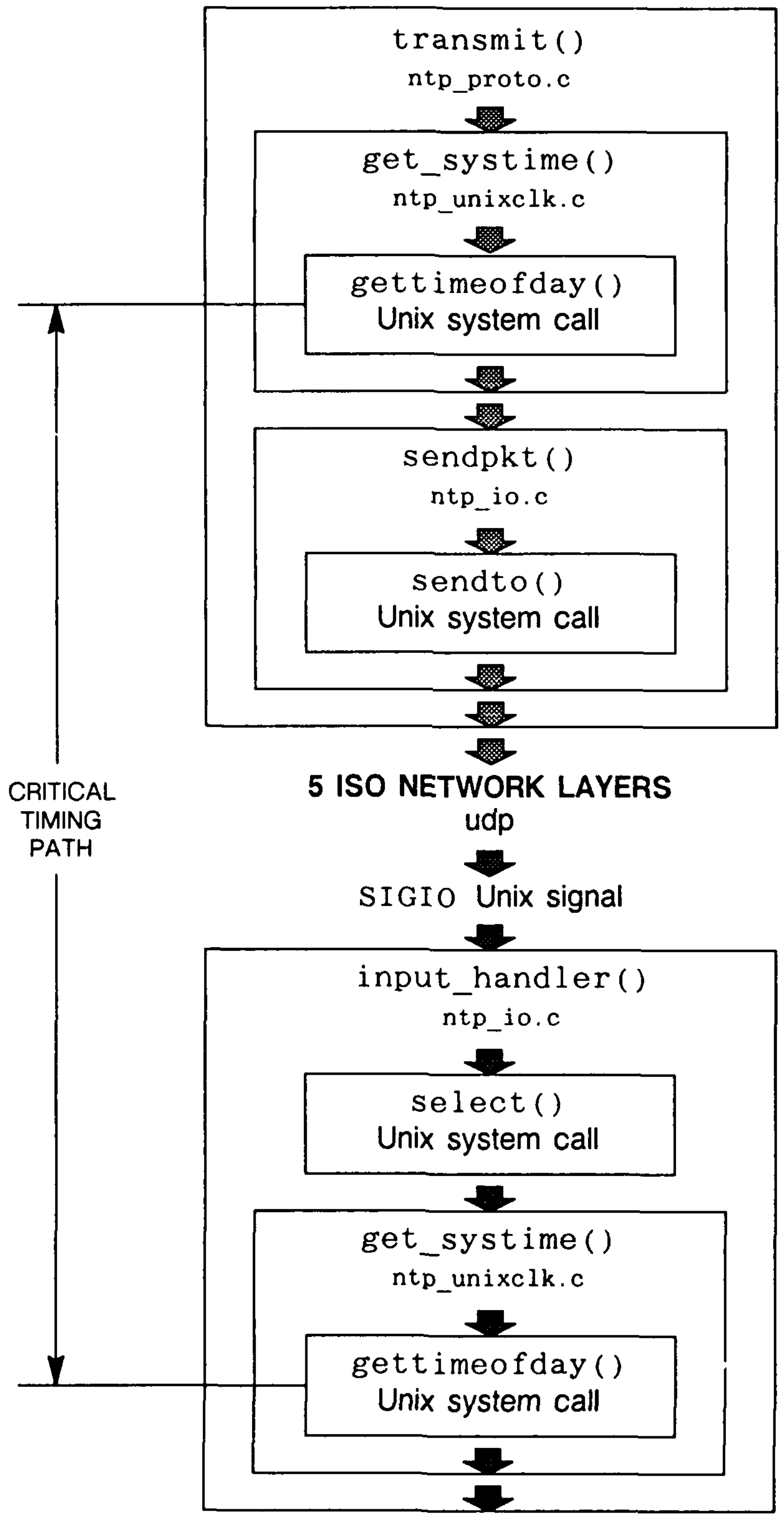


Figure 4. Network Time Protocol implementation (xntpd) critical path.

Unix-oriented Network Time Protocol daemon implementation called `xnptd` [15]. Minor variations in software delays are important but not critical because they are overshadowed by the considerable accuracy lost due to variations in the propagation delay through the wide area network. These delay variations are compensated over time through statistical filtering. In a local area network environment, on the other hand, variations in propagation delay can be controlled more tightly. Hardware implementation provides more accurate and robust realtime clock synchronization.

1.4. IEEE 802.5 STROBE PROPAGATION DELAYS

The transmission rate of all the nodes in an IEEE 802.5 local area network ring is ultimately derived from an oscillator in a single node called the active monitor. Nodes other than the active monitor derive their transmission rate through phase lock loop synchronization with their receive signal. Due to the nature of phase lock loops, there may be instantaneous variation between the reception rate and the transmission rate. This variation is called jitter. The average rate, however, is that of the oscillator in the active monitor.

IEEE 802.5 employs four types of encoded symbols. They are designated 1, 0, J, and K [16]. Unique representations [17] for the four symbol types requires selection of binary states during two half-symbol periods, called unit intervals. Clocking the node internal circuitry at the half-symbol or unit-interval rate, rather than at the symbol rate, simplifies data encoding and decoding. This implies that the actual frequency of the oscillator in the active monitor, and the nominal frequency of the receive clock recovered by the phase lock loops in all the nodes, is twice the symbol rate, or 8 megahertz for the 4-megahertz bus and 32 megahertz for the 16-megahertz bus.

In addition to the oscillator, the active monitor contains the latency buffer [18] of the ring. The latency buffer intentionally delays the active monitor receive signal before retransmitting it around the ring. As shown in figure 5, the latency buffer can be partitioned into two components, the circulation buffer [19] and the elastic buffer [20].

The circulation buffer is a serial-input-serial-output shift register. It ensures that there is sufficient ring latency to guarantee that all 24 symbols (48 unit intervals) forming a token

[15] The figure was derived through examination of the source code listing for Version 1.3 of the `xnptd` Network Time Protocol daemon obtained via "anonymous ftp" from the University of Delaware.

[16] See IEEE 802.5–1989, sec. 5.1, pp. 65–67.

[17] The representations are unique in the sense that the next bit can be uniquely determined given the preceding bit. This qualification is due to differential Manchester encoding.

[18] See IEEE 802.5–1989, sec. 3.7, pp. 41–42, and sec. 5.5, pp. 67–68.

[19] The term "circulation buffer" is the author's own creation.

[20] See IEEE 802.5–1989, sec. 5.5, p. 68.

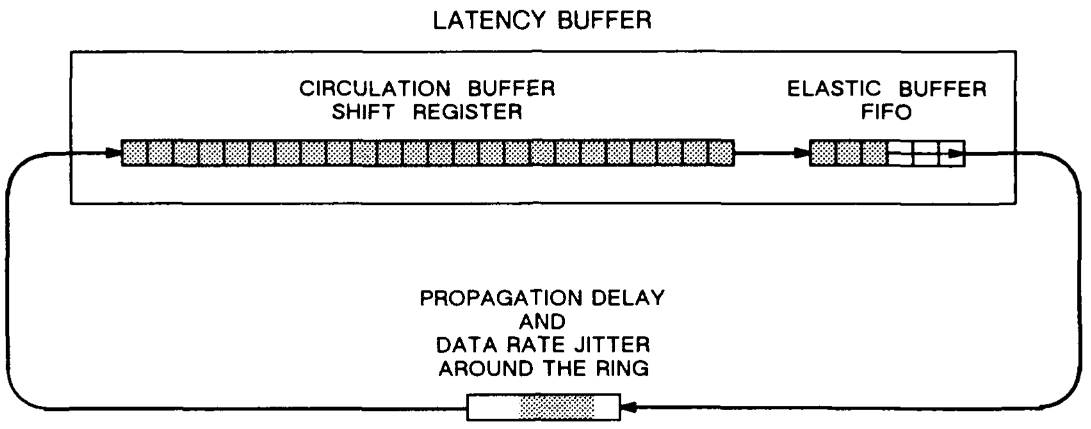

LATENCY BUFFER
CIRCULATION BUFFER SHIFT REGISTER
ELASTIC BUFFER FIFO
PROPAGATION DELAY AND DATA RATE JITTER AROUND THE RING

Figure 5. IEEE 802.5 latency buffer.

protocol data unit can circulate, without the symbols at the beginning of the sequence overwriting the symbols at the end of the sequence, when all nodes are in the repeat mode. The time required to transmit a symbol is 250 nanoseconds for the 4-megahertz bus and 62.5 nanoseconds for the 16-megahertz bus. The delay through the 24-symbol circulation buffer is 6 microseconds for the 4-megahertz bus and 1.5 microseconds for the 16-megahertz bus.

The elastic buffer is a first-in-first-out (FIFO) memory. The elastic buffer compensates for jitter accumulated by the signal as it propagates from the source of its clock at the active monitor, through the nodes around the ring, and finally back to the active monitor. The elastic buffer is initially filled to half its capacity. Data are entered into the elastic buffer by the receive clock derived from the phase lock loop. Data are removed from the elastic buffer by the transmit clock derived from the oscillator. The delay provided by the elastic buffer expands or contracts, as needed, to compensate for the active monitor receive signal being slightly ahead or slightly behind synchronization with the active monitor oscillator.

The 4-megahertz bus requires an elastic buffer with an initial half-capacity delay of three symbols and a range of zero to six symbols [21]. The total delay through the 4-megahertz latency buffer is thus 27 symbols plus or minus 3 symbols, or 6.75 microseconds plus or minus 0.75 microsecond. The 16-megahertz bus requires an elastic buffer with an initial half-capacity delay of 16 symbols and a range of 0 to 32 symbols. The total delay through the 16-megahertz latency buffer is thus 40 symbols plus or minus 16 symbols, or 2.5 microseconds plus or minus 1 microsecond.

[21] See IEEE 802.5-1989, sec. 5.5, p. 68.

These figures represent the minimum delays through the active monitor. The delays may be longer through a particular implementation. The Texas Instruments TMS380C16, for example, provides an elastic buffer with a range of 0 to 15 unit intervals, or 0 to 1.875 microseconds for the 4-megahertz bus, and a range of 0 to 63 unit intervals, or 0 to 1.9675 microseconds for the 16-megahertz bus [22].

The retransmission delay through nodes built using the Texas Instruments TMS380 family of integrated circuits consists of a clocked delay through a known series of internal flip-flops, plus additional logic propagation delay through the receive and transmit circuitry [23]. The clocked delay for both the "first-generation" and the "second-generation" components is 4.5 unit intervals. A unit interval can be split because the clock signal has both a rising and a falling edge. The 4.5 unit-interval clocked delay for the 4-megahertz bus is 562.5 nanoseconds and for the 16-megahertz bus is 140.6 nanoseconds. Texas Instruments has not characterized the additional logic propagation delay through the receive and transmit circuitry. They estimate, but do not guarantee, that the delay is between 80 and 200 nanoseconds for the "first-generation" components and between 50 and 150 nanosceonds for the "second-generation" components.

Strobe resolution over the IEEE 802.5 local area network cannot be better than the local area network jitter. The worst-case jitter is plus or minus 0.75 microsecond for the 4-megahertz bus and plus or minus 1 microsecond for the 16-megahertz bus. If the location of the active monitor at the time of strobe transmission is not known to the processor software, strobe resolution cannot be better than plus or minus half the maximum latency buffer delay, or plus or minus 3.75 microseconds for the 4-megahertz bus and plus or minus 1.75 microseconds for the 16-megahertz bus [24]. These latter figures include the effects of jitter. If nothing is known by the processor software about the current local area network ring configuration at the time of strobe transmission, strobe resolution is only slightly better than plus or minus half the maximum possible propagation time around the ring. The propagation time around the ring is called the ring latency.

There is overhead associated with determining the ring configuration. If a strobe resolution of plus or minus half the maximum ring latency is good enough, then ignoring ring configuration is the preferred approach.

1.5. IEEE 802.5 RING CONFIGURATION

The IEEE 802.5 media access control sublayer provides a mechanism for determining the ring configuration. Each node maintains the address of its nearest upstream neighbor

[22] See Texas Instruments, *TMS380 Second-Generation Token Ring*, 1990, sec. 2.3.3, p 2–10.

[23] See Appendix A.

[24] For more on jitter, see IEEE 802.5–1989, sec. 7.5.3, pp. 86–88.

(UNA) currently inserted within the ring. The upstream neighbor's address is captured as follows. The last octet of a frame protocol data unit is the frame status field [25]. The frame status field contains two identical copies of a symbol called the address recognized bit. The address recognized bit is initialized as zero by the node originating the frame. Any node recognizing the frame destination address as one of its own addresses retransmits the bit as a one as the frame passes through the node. If the bit enters the node as a zero and exits as a one, the node knows that it is the first node to recognize the destination address. By broadcasting a frame with a destination address that all nodes recognize, the first node to recognize the destination address is the first node inserted within the ring downstream from the node that originated the frame. The first node to recognize the destination address can then capture its upstream neighbor's address by copying the frame source address field.

The act of inserting or removing a node from the ring, effected by the trunk coupling unit [26], causes a temporary break in the transmission path through the media [27]. The node detecting the break on its receive side, called the beacon node, transmits a "beacon" media access control frame to inform nodes downstream of the break and its location. When media connection is reestablished, the active monitor broadcasts to all nodes the "active monitor present" media access control frame [28]. The first node downstream from the active monitor locally records its upstream neighbor's address, which is the address of the active monitor. The node then broadcasts to all nodes a "standby monitor present" media access control frame. This allows the next node downstream to obtain its upstream neighbor's address. The process continues until all nodes have obtained their upstream neighbors' addresses.

The media access control sublayer software contains a component, called the configuration report server, which maintains the address of the active monitor and the address of the upstream neighbor of each node [29]. The configuration report server obtains this information by monitoring the broadcast of "report new active monitor" [30] and "report stored upstream neighbor's address" [31] media access control frames. Software can reconstruct the ring configuration by tracing the backward chain of upstream neighbors' addresses.

Unfortunately, the configuration report server indicates the current ring configuration, not necessarily the ring configuration at the time of strobe transmission. There is no simple way to sample the ring configuration at the time of strobe transmission. One could include ring configuration data as part of the strobe information field when the software creates the

[25] See IEEE 802.5-1989, sec. 3.2.8, pp. 31-32.

[26] See IEEE 802.5-1989, sec. 1.2, p. 16; sec. 7.4, pp. 82-84.

[27] See IEEE 802.5-1989, sec. 7.4.2, p. 84.

[28] See IEEE 802.5-1989, sec. 4.1.6, p. 48.

[29] See IEEE 802.5-1989, sec. 1.2, p. 14; sec. 2, pp. 21-22; sec. 3.2.4.1, p. 28; sec. 4.1.9, p. 49.

[30] See IEEE 802.5-1989, sec. 3.3.1.14, p. 34.

[31] See IEEE 802.5-1989, sec. 3.3.1.18, p. 34.

strobe, but there still is no guarantee those data will match the actual ring configuration when the strobe reaches the local area network media. Furthermore, collecting and packing ring configuration data for every strobe creates considerable overhead, which is wasteful when one considers that the data are seldom used.

Rather than trying to capture the ring configuration at the time of strobe transmission, it is far easier to simply invalidate strobes that occur close to changes in ring configuration. Before processing a realtime clock sample value, the realtime clock synchronization software checks to see if the local area network has recently undergone, or currently is undergoing reconfiguration. Due to delays in updating the configuration report server internal data, there is a possible race condition in which the realtime clock synchronization software processes a strobe sample without being aware that the actual network configuration has changed. The impact of the race is minimal, however, since it is quickly caught and little damage can accumulate in the realtime clock synchronization during the short period of time before it is caught. When the processor receives notification from the local area network software of a ring reconfiguration or fault, it calls the realtime clock synchronization software so that the realtime clock synchronization software can detect the race condition, and if necessary, correct the damage.

1.6. FDDI STROBE PROPAGATION DELAYS

FDDI defines 25 types of symbols [32], of which 16 are used to represent 4-bit numeric data and the remaining nine are control symbols. Each symbol is encoded as 5 code bits [33]. FDDI uses the term "code bit" for what IEEE 802.5 calls the binary value within a unit interval. Protocol data units are defined in terms of octets. Every FDDI octet consists of two symbols. The gap between protocol data units contains idle control symbols. The number of idle symbols in the gap may be either even or odd.

Each FDDI node uses its own oscillator [34] to clock its transmit signal and its own phase lock loop to synchronize with its receive signal. The FDDI oscillator frequency is 125 megahertz. Since 4-bit numeric data is encoded as 5 code bits, the burst data rate is four-fifths of 125 megahertz, or 100 megahertz.

Each node must compensate for jitter between its local oscillator frequency and its phase lock loop frequency. This differs from IEEE 802.5, where jitter compensation is only necessary in a single node called the active monitor. The use of an independent oscillator at each

[32] See ANSI X3T9.5/88-148, *FDDI Physical Layer Protocol (PHY-2)*, sec. 3.35, p. 6.; sec. 7.2, p. 17; symbol encoding table p. 23.

[33] See ANSI X3T9.5/88-148, *FDDI Physical Layer Protocol (PHY-2)*, sec. 3.2, p. 4.; sec. 7.1.1, p. 16.

[34] See ANSI X3T9.5/88-148, *FDDI Physical Layer Protocol (PHY-2)*, sec. 8.1.2, p. 26.

node, as shown below, greatly complicates FDDI jitter compensation in comparison to that for IEEE 802.5.

FDDI has two components for jitter compensation, the elasticity buffer [35] and the smoother [36]. The fixed-delay function, performed by the circulation buffer in IEEE 802.5, is incorporated within the smoother in FDDI. The variable-delay function, performed by the elastic buffer in IEEE 802.5, is performed at the code bit level in the elasticity buffer and at the symbol level by the smoother in FDDI.

The elasticity buffer is a first-in-first-out memory used as a variable-length shift register [37] to delay the local area network code bits propagating through it. The elasticity buffer is initialized to a delay of half its capacity. The delay expands or contracts, as needed, to compensate for differences between the phase lock loop frequency derived from the receive signal and the local oscillator frequency used for the transmit signal. The delay is adjusted, as necessary, during the gap between protocol data units. If the delay is greater than half the elasticity buffer capacity and expanding, the adjustment removes an idle symbol from the gap. If the delay is less than half the elasticity buffer capacity and contracting, the adjustment inserts an additional idle symbol into the gap.

An elasticity buffer implementation that inserts or deletes symbols in pairs rather than one at a time is also permitted [38]. Such an implementation inserts and deletes symbol pairs half as frequently and requires twice the capacity compared to one that inserts and deletes symbols individually.

The FDDI specification places limits on the oscillator drift and on the length of protocol data units. Assuming that the elasticity buffer is properly adjusted immediately before the protocol data unit, these limits prevent the elasticity buffer from expanding or contracting beyond its capacity in the middle of a protocol data unit. The required elasticity buffer capacity is computed as follows [39]. The maximum frame length is 9000 symbols [40]. At 5 code bits per symbol, this is equivalent to 45,000 code bits. Local oscillators are required to

[35] See ANSI X3T9.5/88-148, *FDDI Physical Layer Protocol (PHY-2)*, sec. 8.1.2, p. 26; sec. 8.6, pp. 29-31.

[36] See ANSI X3T9.5/88-148, *FDDI Physical Layer Protocol (PHY-2)*, sec. 8.8, pp. 32-40.

[37] The elasticity buffer can be described alternatively as a circular queue whose input and output pointers are counters clocked by the receive and transmit signal clocks, respectively. For implementation details, see Jerry D. Hutchison, Christopher Baldwin, and Bruce W. Thompson, "Development of the FDDI Physical Layer," *Digital Technology Journal*, v. 3, n. 2, Spring 1991, pp. 22-24.

[38] See ANSI X3T9.5/88-148, *FDDI Physical Layer Protocol (PHY-2)*, sec. 8.8.1, p. 34.

[39] See ANSI X3T9.5/88-148, *FDDI Physical Layer Protocol (PHY-2)*, sec. 8.6, p. 29.

[40] See ANSI X3T9.5/88-139, *FDDI Media Access Control (MAC-2)*, sec. 7.2.3, p. 33. A typo in the heading identifies this as sec. 7.2.1.

have a drift no greater than plus or minus 50 parts per million, or 0.005 percent [41]. The maximum drift between two such oscillators occurs when they drift in opposite directions. The maximum drift between oscillators is thus plus or minus 100 parts per million, or 0.01 percent [42]. Taking 0.01 percent of 45,000 code bits yields a drift error of 4.5 code bits. Since the oscillators are independent, there is also a possible phase error between them of plus or minus half an oscillator cycle, or plus or minus 0.5 code bits. Thus the elasticity buffer must support a range of at least plus or minus 5 code bits, which is plus or minus the length of one idle symbol, per gap between protocol data units.

The elasticity buffer compensates for the receive signal frequency of a node being faster than its transmit signal frequency by periodically deleting a symbol, or a pair of symbols, in the gap between protocol data units. As the difference between the frequencies increases, the rate at which symbols are deleted from these gaps also increases. Consider a series of nodes such that each node transmits at a slightly higher frequency than the next downstream node. Each node in the series, except perhaps the first one, has a receive frequency greater than its transmit frequency. A situation can arise where enough error has accumulated independently in each node so that they each decide to delete symbols from the same gap between protocol data units. There is obviously a problem if the number of symbols deleted is greater than the number of symbols in the gap [43].

The purpose of the smoother is to prevent the gap between protocol data units from becoming too small [44]. In basic mode, the smoother seeks to maintain a minimum gap size of 14 symbols [45]. If the gap falls below 14 symbols, the smoother inserts idle symbols. If the gap exceeds 14 symbols, the smoother deletes idle symbols previously inserted. The goal of 14 symbols was selected because the media access control sublayer of a node is not required to copy into the node frames shorter than 12 symbols [46], and an additional pair of symbols may be lost in the elasticity buffer of the next downstream node. Hybrid mode is similar, except that the smoother seeks to maintain a minimum gap size of five symbols.

Both the elasticity buffer and smoother affect the obtainable strobe resolution. The worst-case elasticity buffer situation is just prior to its accumulation of enough error to cause it to insert or delete a symbol or a pair of symbols. If it inserts or deletes one symbol at a time, then the strobe resolution through it is plus or minus one symbol. If it inserts or deletes

[41] See ANSI X3T9.5/88-148, *FDDI Physical Layer Protocol (PHY-2)*, sec. 8.1.2, p. 26.

[42] Ibid.

[43] For statistical simulation results on how often this occurs, see Jerry D. Hutchison, Christopher Baldwin, and Bruce W. Thompson, "Development of the FDDI Physical Layer," *Digital Technology Journal*, v. 3, n. 2, Spring 1991, pp. 24-25.

[44] See ANSI X3T9.5/88-148, *FDDI Physical Layer Protocol (PHY-2)*, sec. 8.8, p. 32.

[45] See ANSI X3T9.5/88-148, *FDDI Physical Layer Protocol (PHY-2)*, sec. 8.8, p. 33.

[46] See ANSI X3T9.5/88-139, *FDDI Media Access Control (MAC-2)*, sec. 7.3.1, p. 33.

a pair of symbols at a time, then the strobe resolution through it is plus or minus two symbols. Since there are 5 code bits per symbol, and each code bit requires 8 nanoseconds to transmit at 125 megahertz, the strobe resolution through the elasticity buffer is plus or minus 40 nanoseconds if it inserts or deletes one symbol at a time, and plus or minus 80 nanoseconds if it inserts or deletes a pair of symbols at a time.

The worst-case situation involving both the smoothers and the elasticity buffers is as follows. Recall the example above where each node in a series of nodes receives at a slightly higher frequency than it transmits. As noted above, it is possible for the elasticity buffers in the nodes to decide, independently of one another, to delete symbols simultaneously at the next gap between protocol data units. Assume this takes place at a gap called gap *A*. Assume, further, that gap *A* contains only the minimum number of symbols. The smoothers maintain the minimum number of symbols in gap *A* by inserting symbols of their own. The protocol data unit following gap *A* passes. The gap after it, call it gap *B*, is very long. The smoothers reclaim the symbols they inserted previously into gap *A* by deleting the same number of symbols from gap *B*. Finally, assume that gap *B* is so long that it allows the elasticity buffers time to get into the same situation that created their simultaneous deletion of symbols from gap *A* in the first place. Now symbols are being deleted from gap *B* by both the elasticity buffers and the smoothers simultaneously. The number of symbols in gap *B* is reduced by the sum of the number of symbols deleted by the elasticity buffers from gap *A* and the number of symbols reclaimed after smoothing gap *A*. In other words, in the worst case the number of symbols is reduced by twice the number deleted by the elasticity buffers.

In addition to the degradation in strobe resolution due to the smoother and elasticity buffer, there is also degradation from the quantization error due to the receive signal being asynchronous with the local oscillator. This error is plus or minus half of a code bit time, which at 125 megahertz is 4 nanoseconds.

The resolution of a strobe as it passes through an FDDI local area network node is degraded by the sum of twice the resolution degradation through the elasticity buffer plus the code bit quantization error. For a node with an elasticity buffer that inserts or deletes symbols individually, the total is 0.084 microseconds. For a node with an elasticity buffer that inserts or deletes symbols in pairs, the total is 0.164 microseconds [47]. These numbers reflect the worst-case situation. Given a large number of nodes, the worst-case situation, while possible, is highly unlikely. The actual strobe resolution may be better represented as a binomial distribution for which the worst-case situation limits the domain of the distribution.

[47] See ANSI X3T9.5/88-148, *FDDI Physical Layer Protocol (PHY-2)*, Annex A, pp. 45-46.

As was the case for IEEE 802.5, FDDI must provide sufficient ring latency for a token protocol data unit to circulate [48]. The smallest possible token contains 5 octets [49]. The 5-octet minimum delay is guaranteed by the requirement that each of the two or more nodes in the ring provide at least 3 octets of delay in repeat mode. Since there are two symbols per octet, 5 code bits per symbol, and 8 nanoseconds of delay per code bit at 125 megahertz, the minimum delay through a node is 0.24 microseconds.

Without knowledge of the ring configuration, the obtainable strobe resolution is slightly better than the ring latency.

[48] See ANSI X3T9.5/88-148, *FDDI Physical Layer Protocol (PHY-2)*, sec. 8.1.3, p. 27.

[49] See ANSI X3T9.5/88-139, *FDDI Media Access Control (MAC-2)*, sec. 7.2.1, p. 32.

2.0 REALTIME CLOCK TOPICS

This section presents miscellaneous adjustable-rate realtime clock topics. These topics apply to both backplane bus and local area network distributed realtime clock synchronization. Due to its length, adjustable-rate realtime clock hardware minimization is presented separately in the section immediately following this section.

2.1. INTERVAL TIMER IMPLEMENTATION METHODS

Interval timers are used to interrupt or release processing at a specific future time. The occurrence of the specific future time is called a time event. Interval timer applications include peripheral driver watchdog timers, time-slice resource scheduling, network communication packet flow control, physical world realtime event activation, and strobe generation for realtime clock synchronization.

Preemptive operating systems typically schedule time events using a single interval timer. Time events are recorded in memory on a queue arranged in the order that they are to occur. The interval timer is assigned to the time event that occurs soonest, which is also the one at the front of the queue. When the time event occurs, the operating system obtains its next time event from the queue, loads the interval timer with the value needed to signal the next time event, and removes the expired time event from the queue. The interval timer is also loaded when the system inserts a new time event occurring sooner than the time event currently using the interval timer. The new time event is also placed at the front of the queue.

Since the time event queue is sorted with the soonest time event at the front, determining the next time event when the current time event expires is very efficient. On the other hand, insertion of a new time event requires a linear search to properly place it within the queue. This is inefficient when the number of time events is large. Employing a heap with the soonest time event on the top [50], or employing separate time events queues for short-term, medium-term, and long-term time events [51], improves efficiency.

Before embarking on the construction of an adjustable-rate interval timer, it is advantageous to consider whether a fixed-rate interval timer suffices. There are two reasons why this may be the case.

[50] See R. E. Barkley and T. P. Lee, "A Heap-Based Callout Implementation to Meet Real-Time Needs," *USENIX Association Conference Proceedings*, June 1988, pp. 213,222.

[51] See G. Varghese and T. Lauch, "Hashed and Hierarchical Timing Wheels: Data Structures for the Efficient Implementation of a Timer Facility," *Proceedings of the Eleventh Symposium on Operating Systems Principles,*" Nov. 1987, pp. 25-38.

First, short-term time events do not allow a fixed-rate interval timer to accumulate much oscillator drift error in a short period of time. Nearly all crystal oscillator specifications, for example, limit drift to within 100 parts per million or better. Assuming no compensation for drift, a 1-millisecond time interval has a maximum error of 100 nanoseconds. This is of the same order of magnitude as a few processor instruction execution times and is generally shorter than the process context switching time. Therefore, the fixed-rate interval timer resolution may be good enough for the short-term time events of many software-intensive applications.

A long-term time event requiring accuracy can be converted into an anticipatory long-term time event not requiring accuracy followed by a short-term time event correcting any accuracy errors. The anticipatory long-term time event gets within range of a sufficiently-accurate short-term time event without passing the desired event time. When the anticipatory long-term time event occurs, the associated processing creates the short-term time event using the accurate time of day to reach the desired event time.

Second, the application, as reflected by its supporting operating system, may not require accurate interval timing. Time-sharing operating systems, such as Unix [52], use the interval timer to partition time into time slices of uniform duration. Time events are processed at time-slice boundaries. Time events have a coarser time granularity than those for preemptive operating systems. There is generally no requirement to make the duration of time slices exact as long as their variation does not cause drift in the time-of-day realtime clock or in the time of occurrence of long-term time events.

Adjustable-rate interval timer hardware implementation depends on the implementation method used for the rest of the adjustable rate realtime clock.

The hidden offset method [53] partitions the clock counter into a most-significant portion visible to the application and a least-significant portion visible only to the realtime clock synchronization algorithm. The interval timer operates in parallel with the most-significant portion of the clock counter. Both the interval timer and the most-significant portion of the clock counter are clocked by the carry output from the least-significant portion of the clock counter. The interval timer decrements at the same time that the most-significant portion of the clock counter increments.

The periodic phase modification method [54] increments the clock counter on each cycle of the phase counter. The interval timer decrements using the same phase counter output

[52] See Maurice J. Bach, *The Design of the UNIX Operating System*, Prentice-Hall, Englewood Cliffs, New Jersey, 1986, pp. 260–264. Also see Samuel J. Leffler, Marshall Kirk McKusick, Michael J. Karels, and John S. Quarterman, *The Design and Implementation of the 4.3BSD Unix Operating System*, Addison-Wesley, Reading, Massachusetts, 1989, pp. 50–53.

[53] See Wilcox, NOSC TR 1400, pp. 9–12.

signal used to increment the clock counter. This is the same approach used to implement the rate-adjustment counter [55]. The rate-adjustment counter is, in reality, a special-purpose interval timer dedicated to controlling the clock itself.

The phase accumulation method [56] offers two alternatives. One could implement the interval timer function as an accumulator similar to the one used for the clock value. A less costly and slightly less accurate approach is to implement the interval timer as a simple counter that counts in the units of tick periods. In the latter case, software converts the desired time interval into an equivalent number of tick periods by multiplying the time interval by the inverse of the tick period. While this approach does not take into account variation in the tick period after initialization of the interval counter, the impact of such variation is minor over a short time interval since there is insufficient time for much error to accumulate.

2.2. RATIONALE FOR THE PHASE COUNTER

The periodic phase modification method may seem unnecessarily complex [57]. Consider an alternative method called the tick modification method. It differs from the phase modification method in that the tick modification method does not define a phase counter between the oscillator and the clock counter. In the phase modification method, the clock counter increments at approximately the frequency of the oscillator divided by the number of states in the normal phase counter cycle. In the tick modification method, on the other hand, the clock counter increments at approximately the same frequency as the oscillator.

As is the case for the periodic phase modification method, the tick modification method determines whether the local clock counter is running too fast or too slow by processing the sample values captured by consecutive strobes. The tick modification method controls the clock counter rate as follows. Define a tick as the pulse whose rising edge causes the clock counter to increment by one. If the clock counter is running too fast, occasionally it deletes a periodic oscillator tick. If the clock counter is running too slow, occasionally it inserts an additional tick between two periodic oscillator ticks.

The tick modification method seems to avoid the notion of phase altogether. Unfortunately, the hardware implementation of the tick modification method is not as simple as it may first appear. Figure 6 shows the tick signal waveform that increments the clock counter for both the case of deleting a tick and of inserting a tick. There is no difficulty in deleting a

[54] See Wilcox, NOSC TR 1400, pp. 12–19.

[55] See Wilcox, NOSC TR 1400, p. 15.

[56] See Wilcox, NOSC TR 1400, pp. 19–20.

[57] The necessity of a phase counter was first challenged within a negative ballot to an IEEE 896.3 draft in the Spring of 1991.

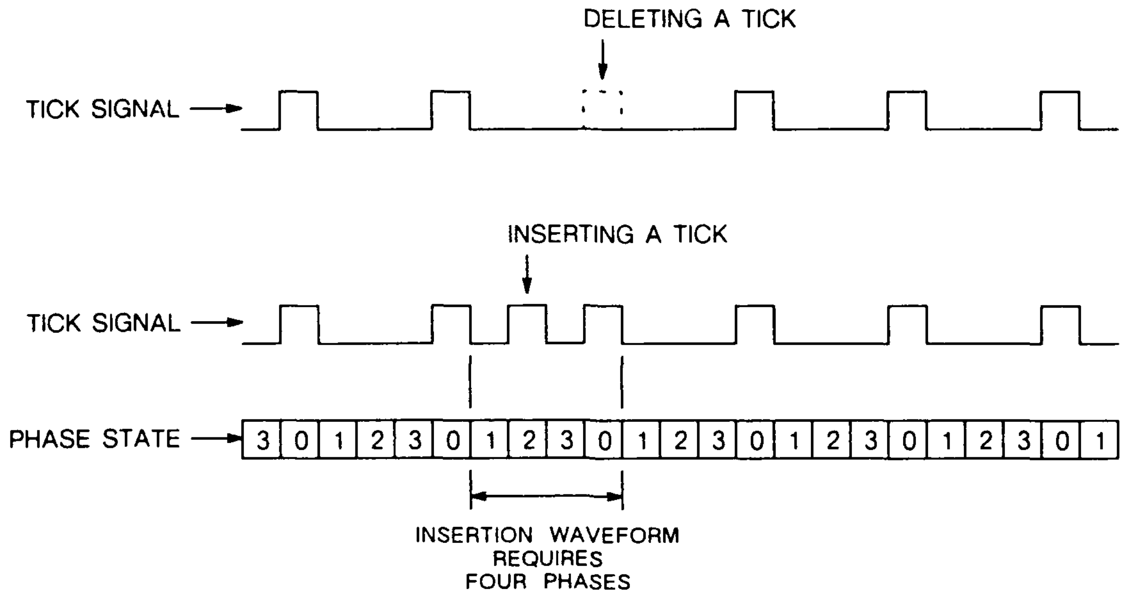


Figure 6. Tick modification method waveforms and phases.

tick; the problem is in inserting a tick. Inserting a tick means placing an additional tick pulse entirely between two existing periodic tick pulses. The additional pulse must be *entirely* between the periodic pulses because that is the only way to generate two rising edges, one for the inserted tick and one for the periodic tick. Any overlap of the pulses generates only one rising edge. The two pulses divide the tick signal cycle into four time intervals. They are

- the time **before** the **inserted tick** rising edge when the tick signal is **low**,
- the time **after** the **inserted tick** rising edge when the tick signal is **high**,
- the time **before** the **periodic tick** rising edge when the tick signal is **low**, and
- the time **after** the **periodic tick** rising edge when the tick signal is **high**.

The four time intervals may differ in duration. Since phase is defined as a fractional part of a periodic signal cycle, the four time intervals are, by definition, four phases of the tick signal cycle. Therefore, the tick modification method does not avoid the concept of phase. Furthermore, the only way for hardware to guarantee the generation of four unique sequential states is for it to have a four-state sequential state machine of some sort. Whatever one chooses to call it, that state machine serves the same purpose as the phase counter.

The rate adjustment resolution of the phase modification method is better than that for the tick modification method. The phase modification method inserts or deletes a single phase from the cycle of the phase counter during modified cycles. As can be seen from figure 6, the tick modification method, on the other hand, makes a full cycle adjustment, consisting of at least four phases, when it inserts or deletes a tick.

2.3. SOFTWARE READ APPROACH ATOMIC ACCESS

Processor accesses to the clock counter must be atomic when the clock counter is wider than the processor access path to it [58]. Figure 7 shows an example of the consequences of a processor read access that is not atomic. The clock counter is 64 bits wide and the processor access path to it is only 32 bits wide. The processor needs two accesses, one to the most-significant half and one to the least-significant half, to obtain the entire contents of clock counter. The clock increments the clock counter independently of the processor. Periodically the increment causes the propagation of a carry from the least-significant half into the most-significant half of the clock counter. When this occurs between the time that the processor reads the most-significant half and the time that the processor reads the least-significant half, the processor obtains an invalid clock value. It is invalid because it is a mix of portions from two different clock values. The most-significant half reflects the clock value before the carry propagation and the least-significant half reflects the clock value after the carry. The error is large, as the example illustrates. The problem exists regardless of which

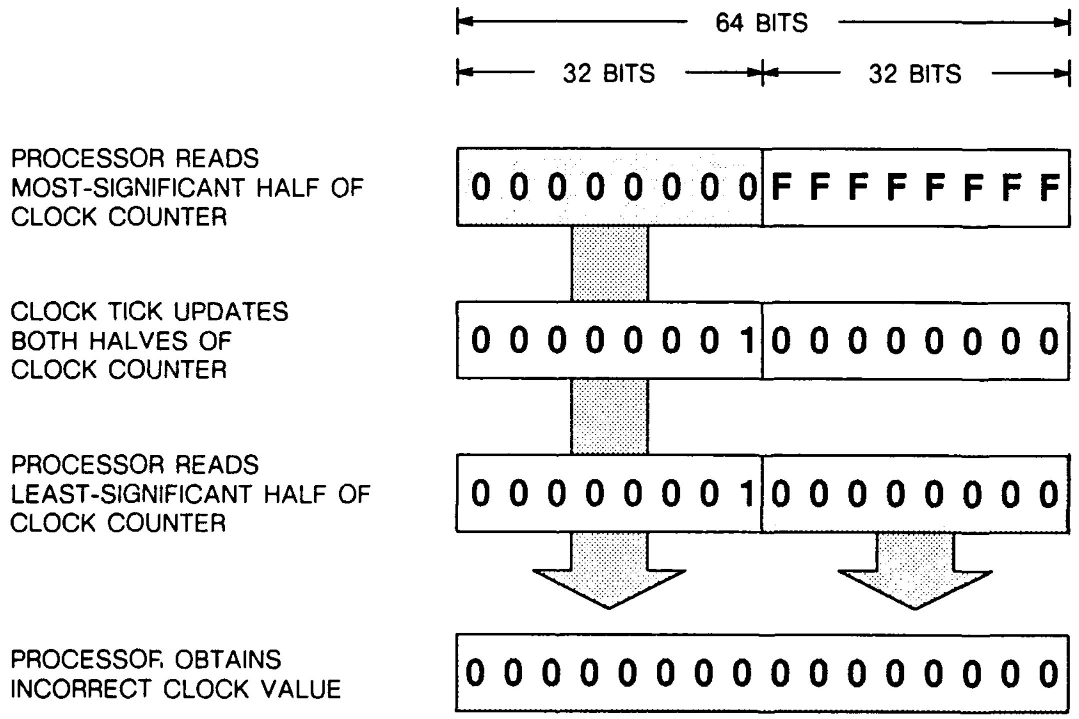

64 BITS
32 BITS
32 BITS
PROCESSOR READS MOST-SIGNIFICANT HALF OF CLOCK COUNTER
0 0 0 0 0 0 0 0 F F F F F F F F
CLOCK TICK UPDATES BOTH HALVES OF CLOCK COUNTER
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
PROCESSOR READS LEAST-SIGNIFICANT HALF OF CLOCK COUNTER
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
PROCESSOR OBTAINS INCORRECT CLOCK VALUE
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 7. Clock counter atomic read access problem.

[58] See Wilcox, NOSC TR 1400, pp. 21-26.

half of the clock counter the processor accessed first, and regardless of whether the processor access is a read or a write.

The software read approach [59] is a popular solution for processor atomic read access. The software makes three accesses rather than two. First the most-significant portion of the clock counter is read, then the least-significant portion is read, and finally, the most-significant portion is read again. It is assumed that all three reads take place in a time interval shorter than half the length of time between successive internal carry propagations from the least-significant portion to the most-significant portion. The software compares the first access with the third access. If they returned the same value, which is almost always the case, then no carry propagation from the least-significant portion into the most-significant portion took place between the time of the first access and the third access. The clock value is reconstructed by concatenating the most-significant portion read by either the first or the third access to the least-significant portion read by the second access. If, on the other hand, the first and third accesses obtain different values, a carry propagation did take place. There are a number of alternative methods for reconstructing the clock value in that case.

The simplest reconstruction method concatenates least-significant zero bits to the most-significant portion of the clock value read by the third access. This method is based on two assumptions. First, it assumes that the effects of carry propagation are *always* included in the reconstructed clock value [60] because the third processor access reflects the situation *after* any carry propagation between the first and the third processor access. Second, it assumes that the period between the first and the third processor accesses is shorter than the period between clock ticks. A tick that generates carry propagation from the least-significant portion into the most-significant portion of the clock counter leaves all zero bits in the least-significant portion and increments the most-significant portion. The use of zero bits in the reconstructed clock value represents the correct clock value if no clock ticks occur after the one that generated the carry. This is ensured by requiring that there be no more than one clock tick between the first processor access and the third processor access, namely, the tick that potentially generates the carry.

Two other methods for reconstructing the clock value overcome the limitations of the two assumptions above.

The first method makes a fourth processor access, this time to the least-significant portion of the clock counter. It reconstructs the clock value by concatenating the values read by the third and fourth accesses, and then corrects the result by subtracting the execution time

[59] See Wilcox, NOSC TR 1400, pp. 21–23, 25–26.

[60] One can also make the opposite assumption, that the effect of carry propagation is *never* included in the reconstructed clock value, by using the first rather than the third processor access and concatenating all one bits rather than all zero bits.

expended by the software over and above what it needs for the case of the first and third accesses producing the same result. This obviously requires the reconstruction to be uninterruptible so that the subtracted execution time is a known constant.

The second method uses the values already obtained from the three original processor accesses of the clock counter rather than performing additional accesses. The most-significant bit of the value read by the second access is examined. If it is a *one*, then carry propagation took place *after* the second access. The clock value is reconstructed by concatenating the values read by the first and second access. If it is a *zero*, then carry propagation took place *before* the second access. The clock value is reconstructed by concatenating the values read by the third and second access.

Up to this point the discussion has only considered the case where all the bits of the clock counter can be uniquely addressed by two processor accesses. There are also situations where more than two accesses are needed. Consider, for example, processor access to a 64-bit clock counter through a 16-bit access path. This requires four processor accesses to address uniquely all the clock counter bits.

Fortunately, cases involving more than two accesses to address uniquely all the clock counter bits can be partitioned into a nested hierarchy of cases where each member of the hierarchy involves direct access to the most-significant bits and atomic access of the next lower member of the hierarchy. For the 16-bit access path to the 64-bit clock counter, the top of the hierarchy consists of direct access of the 16 most-significant bits and atomic access of the remaining 48 bits. The processor first reads the most-significant 16 bits, then, atomically through a series of instructions, the least-significant 48 bits, and finally, the most-significant 16-bits again. It compares the values returned by the first and third accesses of the most-significant bits and acts accordingly as described previously.

Let the bit positions of the 64-bit clock counter example be enumerated from 0 for the least-significant bit position to 63 for the most-significant bit position as shown in figure 8. Then this atomic read of the 64-bit clock counter can be summarized by the following macro definition:

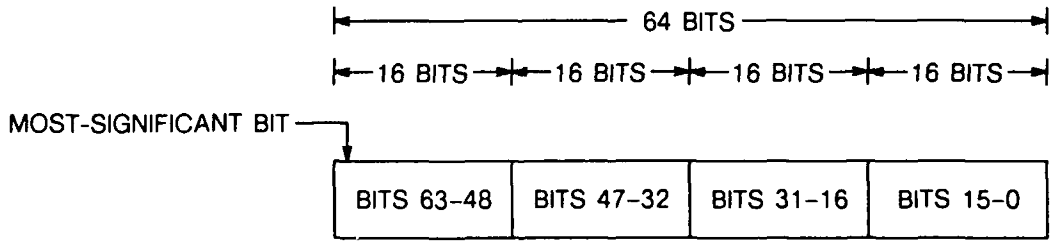


Figure 8. Sixteen-bit access to 64-bit register notation.

```
atomic read bits 63-0 =   read bits 63-48
                              atomic read bits 47-0
                          read bits 63-48 again
                          compare bit 63-48 read values
                          reconstruct bits 63-0
```

Using the same approach, atomic read of the least-significant 48 bits of the clock counter, a component of the macro definition above, can be summarized as follows:

```
atomic read bits 47-0 =   read bits 47-32
                              atomic read bits 31-0
                          read bits 47-32 again
                          compare bit 47-32 read values
                          reconstruct bits 47-0
```

Similarly, atomic read of the least-significant 32 bits of the clock counter, a component of the second macro definition, can be summarized as follows:

```
atomic read bits 32-0 =   read bits 31-16
                              read bits 15-0
                          read bits 31-16 again
                          compare bit 31-15 read values
                          reconstruct bits 31-0
```

Atomic read of the entire 64-bit clock counter, after macro substitution, is defined as follows:

```
atomic read bits 63-0 =   read bits 63-48
                              read bits 47-32
                                  read bits 31-16
                                      read bits 15-0
                                  read bits 31-16 again
                                  compare bit 31-16 read values
                                  reconstruct bits 31-0
                              read bits 47-32 again
                              compare bit 47-32 read values
                              reconstruct bits 47-0
                          read bits 83-47 again
                          compare bit 63-48 read values
                          reconstruct bits 63-0
```

The entire access is atomic if all the comparisons result in equality. Finally, the comparisons are delayed so that the reads can be executed as close to one another as possible.

```
atomic read bits 63-0 =   read bits 63-48
                              read bits 47-32
                                  read bits 31-16
                                      read bits 15-0
                                  read bits 31-16 again
                              read bits 47-32 again
                          read bits 83-47 again
                                  compare bit 31-16 read values
                                  reconstruct bits 31-0
                              compare bit 47-32 read values
                              reconstruct bits 47-0
                          compare bit 63-48 read values
                          reconstruct bits 63-0
```

Note that the read accesses go from the most-significant bits to the least-significant bits sequentially, and then back again to the most-significant bits sequentially. This approach was discovered by Lamport [61]. It is a special case of a general approach for implementing single-writer, multiple-reader shared variables among independent processors without locking [62]. The clock increment hardware is the writer; the one or more processors accessing the clock counter are the readers. The approach does not work for atomic write access.

2.4. RATE ADJUSTMENT RELOAD REGISTER ATOMIC ACCESS

Another area where atomic access is a concern is the rate adjustment reload register of the periodic phase modification method. Processor write access must be atomic when the rate adjustment reload register is wider than the processor access path to it. Figure 9 shows an example of the consequences of a processor write access that is not atomic. The processor needs two accesses, one to the most-significant half and one to the least-significant half, to write the entire contents of rate adjustment reload register. The clock copies the rate adjustment reload count from the rate adjustment reload register to the rate adjustment counter independently of the processor. When this occurs between the time that the processor writes the most-significant half and the time that the processor writes the least-significant half, the rate adjustment counter copies an invalid rate adjustment reload count. It is invalid because it is a mix of portions from two different rate adjustment reload counts.

[61] See Leslie Lamport, "Concurrent Reading and Writing of Clocks," *ACM Transactions on Computer Systems*, v. 8, n. 4, Nov. 1990, pp. 305–310.

[62] See Leslie Lamport, "Concurrent Reading and Writing," *Communications of the ACM*, v. 20, n. 11, Nov. 1977, pp. 806–811. Also see textbook by Mamoru Maekawa, Arthur E. Oldehoeft, and Rodney R. Oldehoeft, *Operating Systems: Advanced Concepts*, Benjamin/Cummings Publ., Menlo Park, CA, 1987, pp. 34–37.

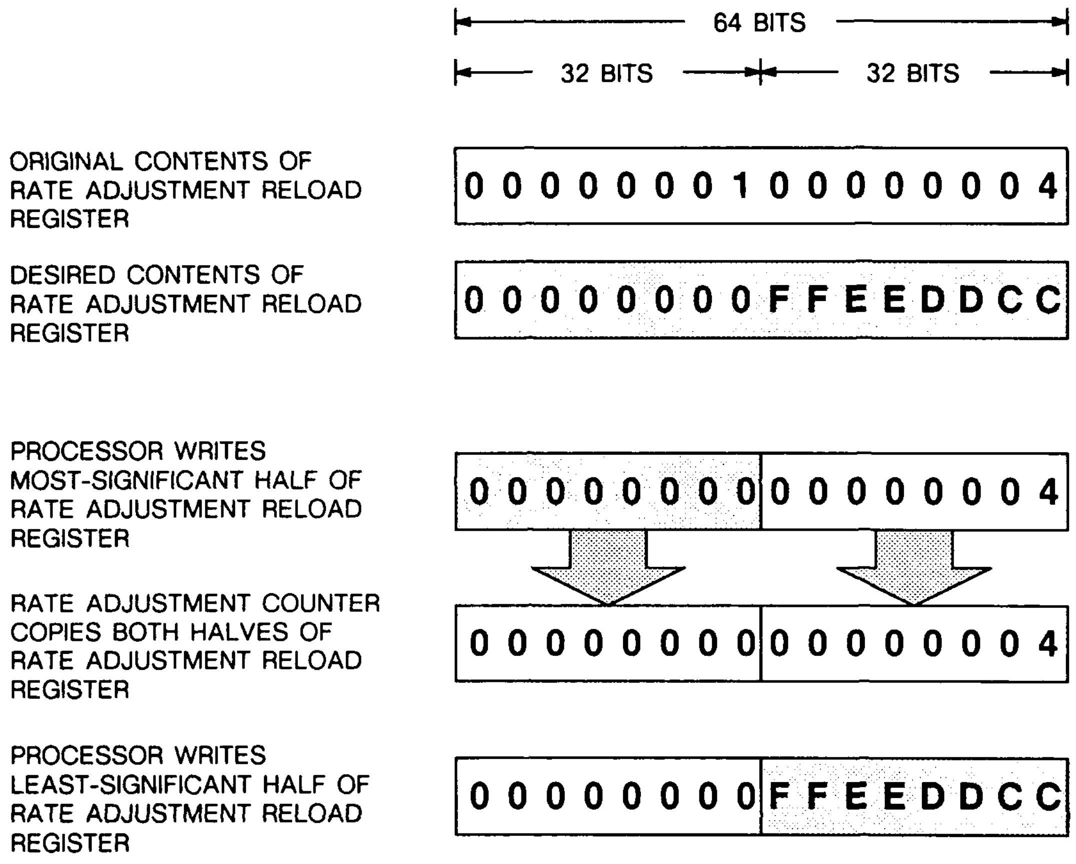

64 BITS
32 BITS
32 BITS
ORIGINAL CONTENTS OF RATE ADJUSTMENT RELOAD REGISTER
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 4
DESIRED CONTENTS OF RATE ADJUSTMENT RELOAD REGISTER
0 0 0 0 0 0 0 0 F F E E D D C C
PROCESSOR WRITES MOST-SIGNIFICANT HALF OF RATE ADJUSTMENT RELOAD REGISTER
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4
RATE ADJUSTMENT COUNTER COPIES BOTH HALVES OF RATE ADJUSTMENT RELOAD REGISTER
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4
PROCESSOR WRITES LEAST-SIGNIFICANT HALF OF RATE ADJUSTMENT RELOAD REGISTER
0 0 0 0 0 0 0 0 F F E E D D C C

Figure 9. Rate adjustment reload register atomic write access problem.

The error can be large, as the example illustrates. The problem exists regardless of which half of the rate adjustment reload register the processor writes first.

The simplest way to provide atomic processor access to the rate adjustment reload register is to disable phase counter modified cycles during processor access. The phase counter modified cycle disable mode is already required to handle infinite rate adjustment periods [63]. This solution requires no additional hardware.

2.5. VALUE-RATE ADJUSTMENT ALGORITHM

The value-rate adjustment algorithm is a local realtime clock synchronization algorithm that uses value adjustment to correct the value error since the last adjustment and rate adjustment in an attempt to prevent accumulation of any further value error at the next adjustment. It is suitable for applications where the *rate of change* of the realtime clock rate is small. Realtime clocks driven by crystal oscillators have this behavior, except perhaps for

[63] See Wilcox, NOSC TR 1400, pp. 15–16.

the first few seconds after power is applied when the crystal oscillator temperature is changing rapidly [64].

Figure 10 shows the value adjustment portion of the value-rate adjustment algorithm in terms of synchronization error versus time. The first adjustment, e_1, simply cancels the value error, E_1, derived from the samples of the first strobe. It assumes that the clocks are incrementing at the same rate. It is seldom the case that they are actually incrementing at the same rate, but that assumption is the best that can be done given samples from only one strobe [65].

Let Sj and Mj be the sample values of the local and master clocks, respectively, captured by strobe j. Let τ be the current local clock value. Sometime after the first strobe, the value of τ is replaced with its new value given by the equation

$$\tau_1 = \tau - E_1 = \tau - (S_1 - M_1)$$

The value of M_1 is saved for use by the next iteration.

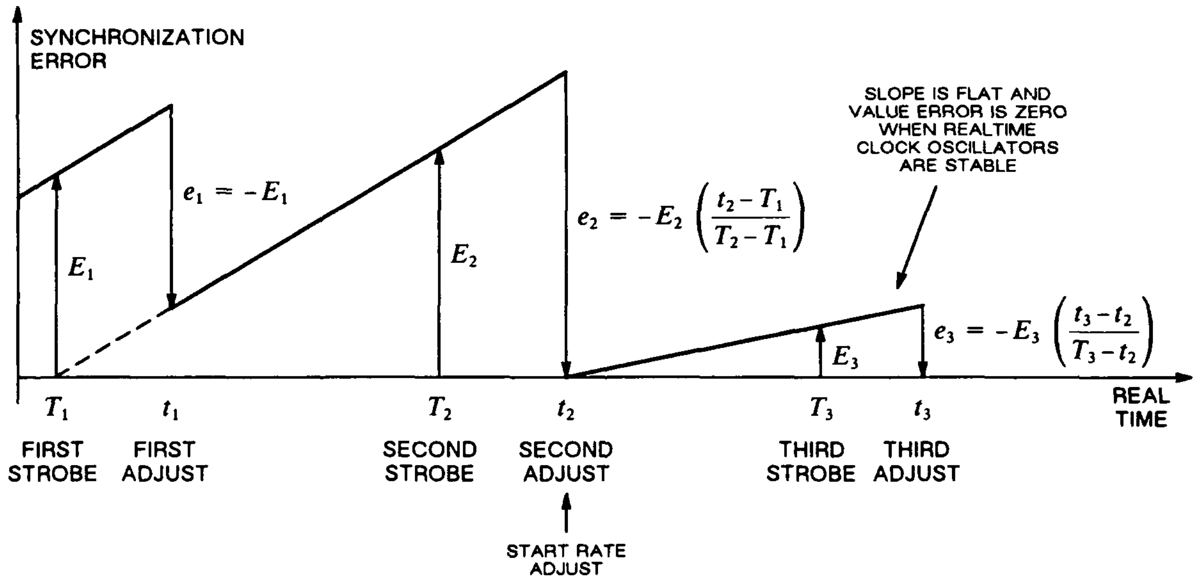

SYNCHRONIZATION ERROR
$e_1 = -E_1$
E_1
E_2
$e_2 = -E_2 \left(\frac{t_2 - T_1}{T_2 - T_1}\right)$
SLOPE IS FLAT AND VALUE ERROR IS ZERO WHEN REALTIME CLOCK OSCILLATORS ARE STABLE
E_3
$e_3 = -E_3 \left(\frac{t_3 - t_2}{T_3 - t_2}\right)$
T_1 FIRST STROBE
t_1 FIRST ADJUST
T_2 SECOND STROBE
t_2 SECOND ADJUST
START RATE ADJUST
T_3 THIRD STROBE
t_3 THIRD ADJUST
REAL TIME

Figure 10. Value-rate adjustment algorithm.

[64] See Wilcox, NOSC TR 1400, pp. 30-32.

[65] See Wilcox, NOSC TR 1400, p. 6.

Once the sample values from the second strobe become available, the rate as well as the value can be adjusted. The second value adjustment, e₂, includes compensation for the rate error. It is given by the equation

$$\tau_2 = \tau - E_2\left(\frac{t_2 - T_1}{T_2 - T_1}\right) = \tau - (S_2 - M_2)\left(\frac{\tau - M_1}{S_2 - M_1}\right)$$

The rate adjustment after the second strobe is given by the equation

$$R_2 = \frac{-E_2}{T_2 - T_1} = -\frac{S_2 - M_2}{M_2 - M_1}$$

The values of τ_2, R_2, and M_2 are saved for use by the next iteration.

The third adjustment, and all adjustments that follow, have the same form. The value adjustment equation is

$$\tau_j = \tau - E_j\left(\frac{t_j - t_{j-1}}{T_j - t_{j-1}}\right) = \tau - (S_j - M_j)\left(\frac{\tau - \tau_{j-1}}{S_j - \tau_{j-1}}\right) \qquad \text{for } j > 2$$

and the rate adjustment equation is

$$R_j = R_{j-1} - \frac{E_j}{T_j - T_{j-1}} = R_{j-1} - \frac{S_j - M_j}{M_j - M_{j-1}} \qquad \text{for } j > 2$$

The values of τ_j, R_j, and M_j are saved for use by the next iteration.

For the periodic phase modification method employing k phases in a normal phase counter cycle, the rate adjustment reload register value is the absolute value of x, where x is given by the equations

$$x_1 = \infty$$

$$x_2 = \frac{1}{k\,R_2} = -\frac{M_2 - M_1}{k\,(S_2 - M_2)}$$

$$x_j = \frac{1}{k\,R_j} = \frac{1}{\dfrac{1}{x_{j-1}} - \dfrac{k\,(S_j - M_j)}{M_j - M_{j-1}}} \qquad \text{for } j > 2$$

If x is negative, then the local clock is running too fast so long modified phase counter cycles are periodically inserted. Otherwise short modified phase counter cycles are periodically inserted.

The evaluation of value adjustment equations requires processor execution time. Since the realtime clock is advancing while the processor is performing the computation, the time between reading the realtime clock value, τ, and replacing it with the updated value, τ_j, should be minimized. This can be accomplished by rewriting the equations in an equivalent form that performs as much of the computation as possible before reading τ. Processor execution time cannot be eliminated entirely. What remains can be compensated by adding a processor execution time correction term, p, to each equation.

$$\tau_1 = \tau - (S_1 - M_1) + p_1$$

$$\tau_2 = \tau \left(\frac{M_2 - M_1}{S_2 - M_1} \right) + M_1 \left(\frac{S_2 - M_2}{S_2 - M_1} \right) + p_2$$

$$\tau_j = \tau \left(\frac{M_j - \tau_{j-1}}{S_j - \tau_{j-1}} \right) + \tau_{j-1} \left(\frac{S_j - M_j}{S_j - \tau_{j-1}} \right) + p \qquad \text{for } j > 2$$

Interrupts, and other forms of processor preemption, must be disabled between the time that the realtime clock is read and the time that it is updated so that the additional term is a constant.

It is also important that the arithmetic instructions employed for the equations have sufficient bits to avoid overflow and truncation problems. Typically this means supporting some form of 64-bit arithmetic. Floating-point instructions may be used where there are no significant instruction execution time data-dependencies affecting the constant p.

Value-rate adjustment avoids the stability problems associated with pure rate adjustment [66]. It also has the advantage of being relatively easy to understand. The disadvantage of value-rate adjustment is that its value adjustment component can make sudden changes in the clock value during periods of rapid change in the clock rate.

[66] See Wilcox, NOSC TR 1400, pp. 36–37.

3.0 REALTIME CLOCK HARDWARE MINIMIZATION

This section presents techniques for minimizing the hardware required to implement the adjustable-rate realtime clock [67]. The periodic phase modification implementation method is assumed.

3.1. ELIMINATING THE CLOCK COUNTER LOAD FUNCTION

The hardware required to implement the adjustable rate realtime clock can be reduced by avoiding implementation of functions easily implemented in software. One such function is software initialization of the *hardware* clock value. If the hardware clock value is not initialized, its value will differ from the value desired by the software by some differential constant. Instead of initializing the hardware clock value, the software determines the value of the constant. The constant is computed by subtracting the actual hardware clock value from the desired software clock value for the same moment in physical time. Once the value of the constant is known, future readings of the hardware clock value can be translated in software by adding this constant to the hardware clock value.

Eliminating the need to load a particular value into a clock counter opens opportunities for implementing the clock counter using fewer, simpler, and smaller integrated circuit components. These are possible because integrated circuit package pins used for counter load data inputs on other counter integrated circuits can be reassigned as counter data outputs on counters without load data inputs. The 74LS393 [68] and the faster 74F393 [69], for example, implement an 8-bit binary ripple [70] counter within a single 14-pin package. The slower 74HC4040 [71] offers even higher density with a 12-bit binary ripple counter within a 16-pin package.

Figure 11 shows a block diagram of a clock counter with attached output and sample registers. The output register is controlled by the processor read logic. It supports

[67] The first adjustable rate realtime clock designed by the author employed the Am2942 integrated circuits as the central element. For Am2942 data, see Advanced Micro Devices, *Bipolar Microprocessor Logic and Interface Data Book*, 1985. The Am2942 appeared to consolidate several clock functional components into a single package. For assignment of Am2942 internal components, see Wilcox, NOSC TR 1400, pp. 17–19. Upon completion, the design was found to require far more interface and control logic than expected. This motivated the present discussion. The problem was not so much the Am2942 as it was separating the functionality that requires hardware implementation from that that does not.

[68] See 74LS393 in Motorola, *FAST and LS TTL Data, 1989*; Signetics, *TTL Data Manual*, 1986, pp. 5-550 – 5-553; Texas Instruments, *The TTL Data Book Volume 2*, 1985.

[69] See 74F393 in Signetics, *FAST Logic Data Handbook*, 1989, pp. 6-422 – 6-426.

[70] Ripple counters have no carry output. Thus the worst case delay from the triggering edge of the clock input on the least-significant integrated circuit to the stable state of the most-significant output on the most-significant integrated circuit may be quite long. For a 32-bit counter, the delay is approx. 160 ns using the 74F393, approx. 240 ns using the 74LS393, and approx. 1 μs using the 74HC4040.

[71] See 74HC4040 in Motorola, *High-Speed CMOS Logic Data*, 1989; Signetics, *High-Speed CMOS Logic Family Data Handbook*, 1991, pp. 805–809; Texas Instruments, *High-Speed CMOS Logic Data Book*, 1987.

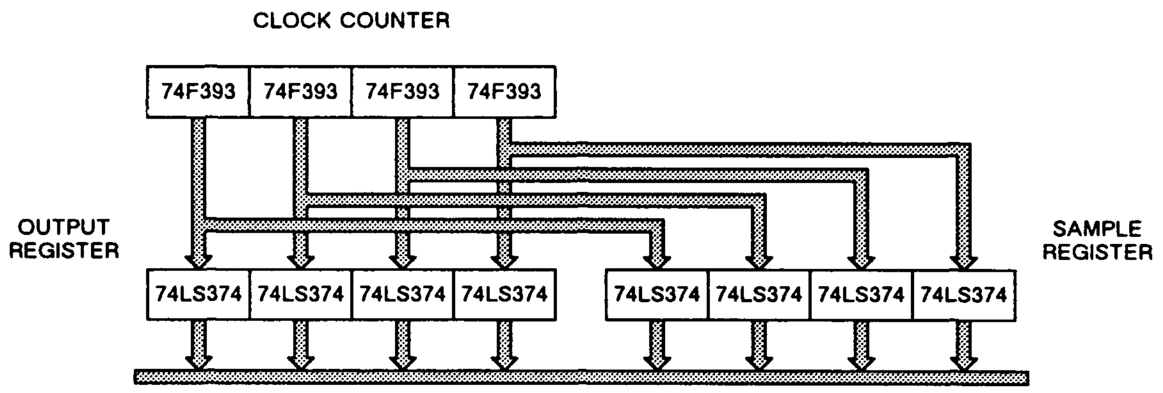


Figure 11. Common clock counter implementation.

resynchronization of the clock counter data transfer when the timing of the clock counter does not match the timing of the processor read, a situation frequently encountered in practical designs. The output register isolates the variation in the delay of the individual ripple counter outputs from the processor bus. The output register can also serve as a read save register for atomic access of the clock counter value by the processor when the access path from the processor is narrower than the width of the clock counter [72]. The sample register is controlled by the strobe detector logic. The three-state enables of the output and sample registers are controlled by the processor address decoder logic.

The number of integrated circuits can be reduced still further by using the 74LS590 integrated circuit [73]. Figure 12 shows a block diagram of the 74LS590. It consists of an 8-bit counter feeding an 8-bit register. The counter clock input is independent of the register clock input. As shown in figure 13, one set of 74LS590 integrated circuits implements the clock counter with output register and another set of 74LS590 integrated circuits implements the clock counter with sample register. The two sets of 74LS590 integrated circuits are forced to maintain the same internal counter value by ensuring that they are always cleared or clocked at the same time and in the same way.

3.2. REDUCING THE SAMPLE REGISTER AND CLOCK COUNTER WIDTH

Another way to reduce hardware through the use of software is to support in the hardware sample register only those least-significant sample value bits that cannot be deduced by the software reading the current clock value after the sample has been taken.

[72] For a discussion of the various atomic access methods, see Wilcox, NOSC TR 1400, pp. 21–27. For a good example of read and write save registers, see MC6840 in Motorola, *8-bit Microprocessor and Peripheral Data*, 1983, where the read and write save registers are designated as the LSB and MSB buffer registers, respectively.

[73] See 74LS590 in Texas Instruments, *The TTL Data Book Volume 2*, 1985.

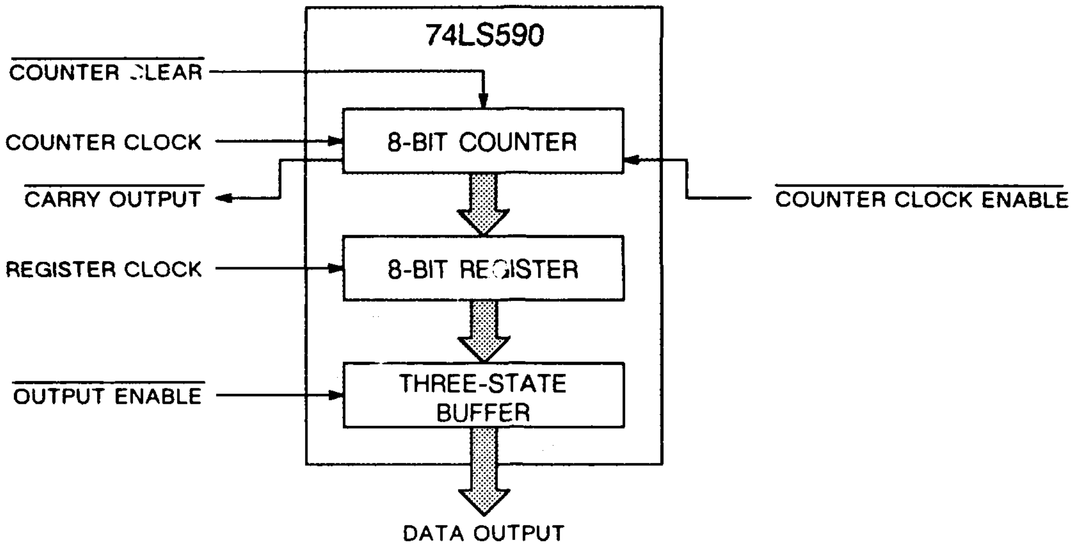


Figure 12. 74LS590 integrated circuit block diagram.

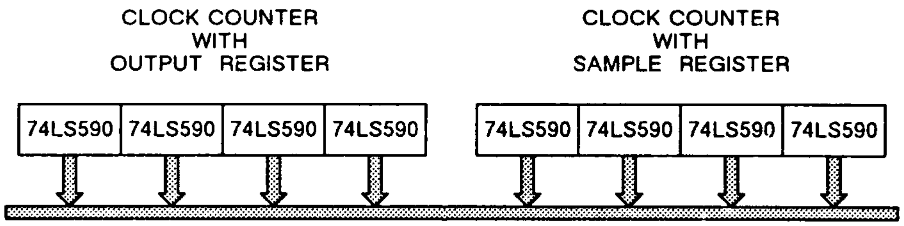

CLOCK COUNTER WITH OUTPUT REGISTER
CLOCK COUNTER WITH SAMPLE REGISTER
74LS590
74LS590
74LS590
74LS590
74LS590
74LS590
74LS590
74LS590

Figure 13. Replicated clock counter implementation.

The software reconstructs each full sample value before the synchronization algorithm processes it. Reconstruction requires two steps. As shown in figure 14, the software first reads and concatenates the most-significant clock value bits, whose numeric significance is not supported by the hardware sample register, to the least-significant bits supplied by the hardware sample register. It then corrects the most-significant bits to compensate for any carry that may have propagated from the clock value least-significant bits to the clock value most-significant bits between the time that the hardware sample register acquired the sample value and the time that the software read the clock value. Carry propagation is detected by comparing the most-significant bit of the hardware sample register to the bit of the same significance in the clock value. If the hardware sample register bit is one and the clock value bit is zero, then a carry has propagated. The effect of the carry is removed by decrementing

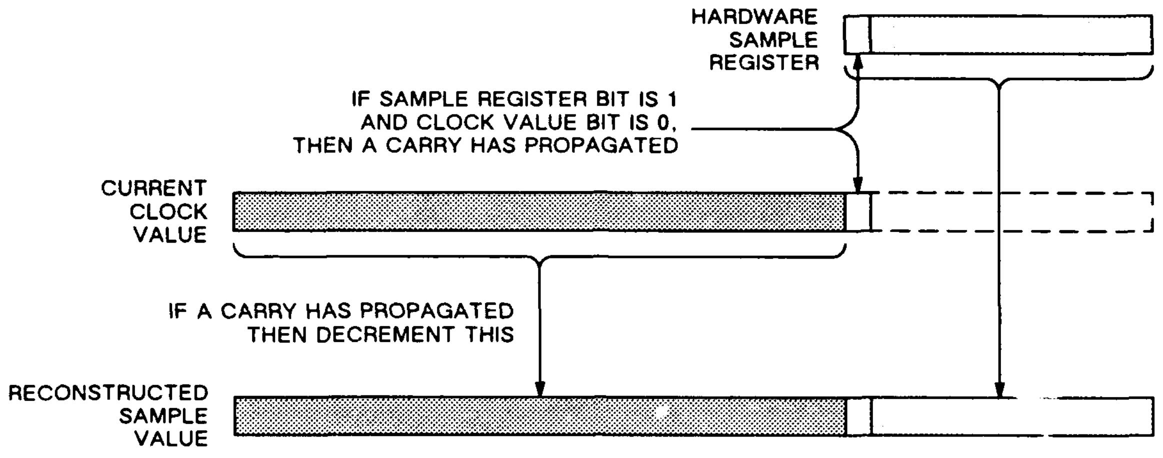

HARDWARE SAMPLE REGISTER
IF SAMPLE REGISTER BIT IS 1 AND CLOCK VALUE BIT IS 0, THEN A CARRY HAS PROPAGATED
CURRENT CLOCK VALUE
IF A CARRY HAS PROPAGATED THEN DECREMENT THIS
RECONSTRUCTED SAMPLE VALUE

Figure 14. Software reconstruction of sample value.

the contents of the most-significant bits that were concatenated to the hardware sample register value.

The minimum width of the hardware sample register is not determined by the maximum expected synchronization error. That determines the minimum width of the full sample value [74]. The minimum width of the hardware sample register is determined rather by the maximum expected delay between the time that the hardware sample register acquires the sample value and the time that the software reads the clock value. If the maximum delay can be represented by an N-bit unsigned number, then the hardware sample register must contain at least N+1 bits. The extra bit is needed to support the detection of carry propagation described above.

A well-known method of reducing the width of a hardware clock counter implements the most-significant clock counter bits in software. When the hardware clock counter overflows, it generates an overflow interrupt informing the software to increment the software portion of the clock counter. The interrupt signal is derived from the hardware clock counter carry output. To avoid atomic access problems, other processing using the full clock counter value must be disabled during this update.

The hardware clock counter width, and the rate it increments, determine the period between hardware clock overflow interrupts. Although often convenient, there is no requirement that the hardware sample register be as wide as the hardware clock counter as long as its sampling requirements, described above, are met.

[74] See Wilcox, NO[illegible] 1400, pp. 29-30.

The role of the software translation constant, introduced in the previous section as a means of eliminating the clock counter load function, can be extended to serve also as the software portion of the clock counter.

3.3. REDUCING THE RATE ADJUSTMENT COUNTER WIDTH

In the periodic phase modification method, the purpose of the rate adjustment counter and associated rate adjustment reload register is to control the periodic insertion or deletion of a phase counter state in order to make minor adjustments in the realtime clock rate [75]. The rate adjustment counter counts downward toward zero. When it reaches zero, it signals the phase counter that an additional phase state should be inserted or deleted from the current cycle of the phase counter. All other cycles of the phase counter have the normal number of phase states. After the count of zero, the rate adjustment counter loads itself with the content of the rate adjustment reload register. It then counts downward repeating the processes just described. The software controls the rate adjustment by specifying the content of the rate adjustment reload register.

The period between modified phase cycles is called the rate adjustment period [76]. Letting k represent the number of phase states in a normal cycle of the phase counter, and letting x represent the rate adjustment counter reload value, the rate adjustment period consists of k times x phase counter phase state periods. The rate adjustment, R, produced by the rate adjustment counter and reload register is defined by the equation

$$R = \frac{1}{k\,x}$$

A perfectly synchronized realtime clock needs a rate adjustment of zero. By solving for x, it is apparent that as the desired rate adjustment approaches zero, the required rate adjustment counter reload value approaches infinity.

$$\lim_{R\to 0} x = \lim_{R\to 0} \frac{1}{k\,R} = \infty$$

This intuitively makes sense since the smaller the desired rate adjustment, the longer the period between insertion or deletion of a phase counter state. Unfortunately, infinity cannot be represented numerically in a finite number of rate adjustment counter bits. The rate adjustment counter is disabled when no rate adjustment is needed.

[75] See Wilcox, NOSC TR 1400 p. 15.

[76] See Wilcox, NOSC TR 1400 p. 14.

Let N represent the number of rate adjustment counter bits. The minimum rate adjustment supported by the counter, $R_{\min}$, is that associated with the maximum number that the counter can represent.

$$R_{\min} = \frac{1}{k\ (2^N - 1)}$$

For example, a 20-bit rate adjustment counter fed by a phase counter with three states per normal cycle supports a minimum rate adjustment of 0.32 parts per million.

Varying the least-significant bits of the rate adjustment counter has little effect on the rate adjustment. This can be seen by comparing the rate adjustment for a count of x to the rate adjustment for a count of $x + \delta$, where δ is some small integer.

$$\Delta R = \frac{1}{k\,x} - \frac{1}{k\ (x + \delta)} = \frac{\delta}{k\ (x^2 + x\,\delta)}$$

Realtime clock synchronization only requires minor rate adjustments. Minor rate adjustments use relatively large values of x. Since x is large, and since its square appears in the denominator, ΔR is very small. Using the example above of a 20-bit rate adjustment counter fed by a phase counter with three states per normal cycle, the difference, ΔR, between the rate adjustment for the largest value of x, which is $2^{20} - 1$, and the next largest value of x, which is $2^{20} - 2$, is only 0.00003 parts per million. This is far less than the minimum rate adjustment, $R_{\min}$, of 0.32 parts per million. Even for much smaller values of x, the value of ΔR is quite small. For example, an x of 2^{12} and an $x + \delta$ of $2^{12} + 1$ gives a ΔR of 0.02 parts per million.

The fact that the least-significant bits of x have little impact on the value of R for all but the smallest values of x can be used to reduce the rate adjustment hardware. Only those most-significant bits of the rate adjustment counter and reload register hardware that have a significant impact need initialization. The least-significant bits of the rate adjustment counter are implemented by a frequency-divider, since software access to their internal content is unnecessary. The least-significant bits of the rate adjustment reload register are eliminated entirely. Figure 15 shows the resulting block diagram.

Let L represent the number of least-significant bits eliminated from the rate adjustment reload register. The rate adjustment reload register is consequently $N - L$ bits wide. The least-significant L bits of the rate adjustment counter are inaccessible by the software. But since the most-significant $N - L$ bits decrement only when the least-significant L bits pass through zero, the software can assume that the least-significant L bits of the rate adjustment counter reload value are zero for the purposes of rate adjustment calculations. Let m represent an arbitrary numeric value for the most-significant $N - L$ bits of the rate adjustment counter reload value. The minimum software-programmable difference between rate adjustment counter reload values is

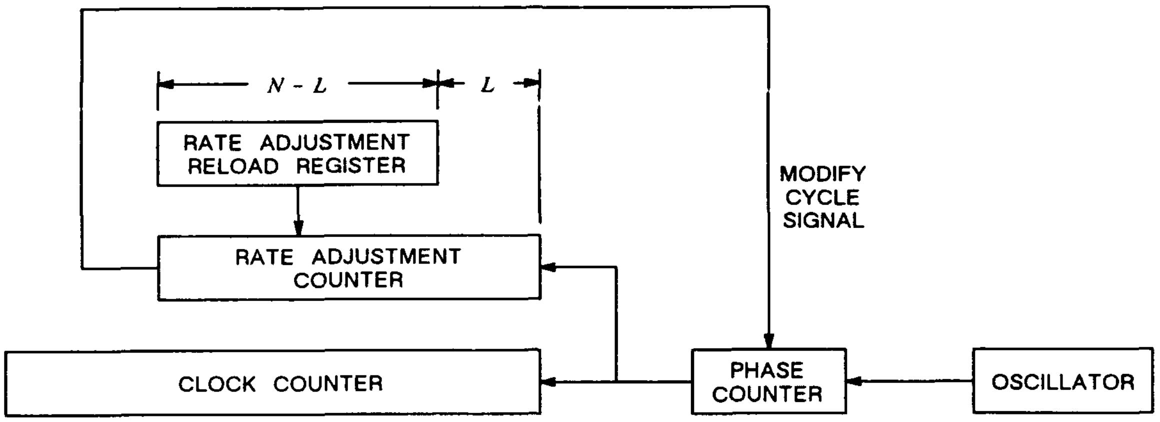


Figure 15. Adjustable-rate realtime clock block diagram.

$$\delta_{\min} = [\,(m+1)\,2^L + 0\,] - [\,m\,2^L + 0\,] = 2^L$$

Substituting the value of $\delta_{\min}$ into the equation for ΔR yields

$$\Delta R = \frac{\delta_{\min}}{k\,(x^2 + x\,\delta_{\min})} = \frac{2^L}{k\,[\,x^2 + 2^L x\,]}$$

With the least-significant L bits now zero, the value of of $R_{\min}$ becomes

$$R_{\min} = \frac{1}{k\,[\,(2^{N-L})\,(2^L) + 0\,]} = \frac{1}{k\,(2^N - 2^L)}$$

There is no point in providing any more rate adjustment accuracy through the inclusion of software-programmable least-significant bits than that supported by the total width of the rate adjustment counter. The two rate adjustment accuracies can be related as follows

$$\Delta R = R_{\min}$$

Figure 16 shows this relationship geometrically. Substituting for ΔR and $R_{\min}$ yields

$$\frac{2^L}{k\,[\,x^2 + 2^L x\,]} = \frac{1}{k\,(2^N - 2^L)}$$

$$x^2 + 2^L x - (2^N - 2^L)\,(2^L) = 0$$

Solving for positive x yields the minimum rate adjustment counter reload value that will give the same rate adjustment accuracy as that associated with $R_{\min}$.

$$x = -2^{L-1} + \sqrt{2^{N+L} - 3 \cdot 2^{2L-2}}$$

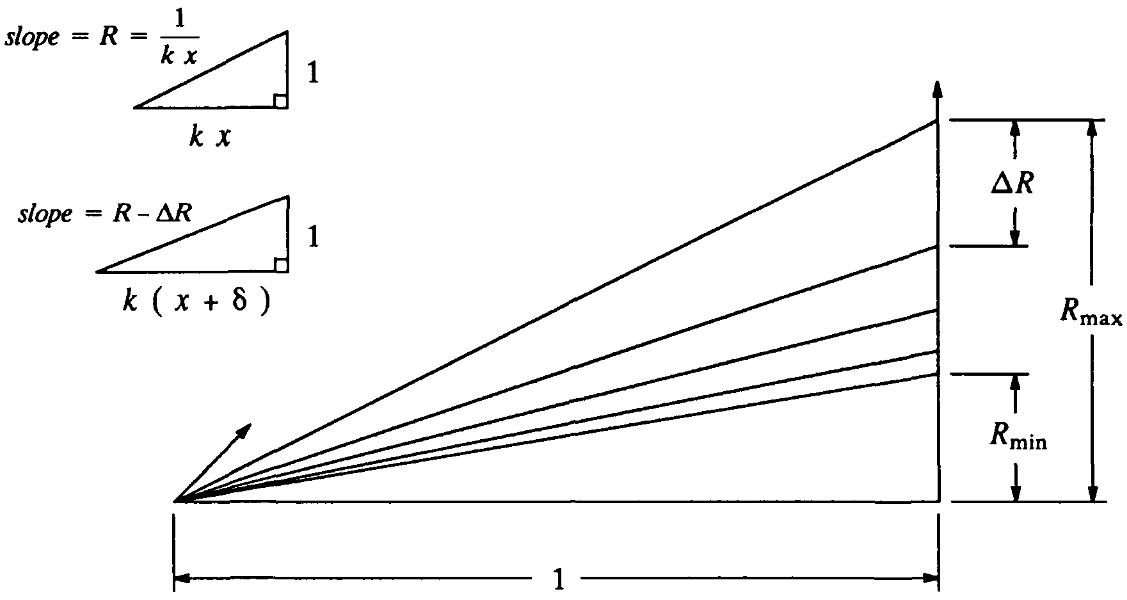


Figure 16. Geometric representation of rate adjustment.

Finally, the rate adjustment corresponding to this value of x, which is the maximum rate adjustment that will give the same rate adjustment accuracy as that associated with $R_{\min}$, is given by the equation

$$R_{\max} = \frac{1}{k\,x} = \frac{1}{k\left(-2^{L-1} + \sqrt{2^{N+L} - 3 \cdot 2^{2L-2}}\right)}$$

Thus, given values for N, L, and k, one can determine the values for $R_{\min}$ and $R_{\max}$.

The design problem is usually the reverse, that is, given $R_{\min}$ and $R_{\max}$, select values for N, L, and k. $R_{\min}$ must be less than or equal to the rate adjustment accuracy required by the application. $R_{\max}$ must be larger than the worst-case drift specified for the crystal oscillator. It must be larger because it must cancel not only the effects of the oscillator drift but also correct minor errors in the clock value. Worst-case drift values commonly encountered in crystal oscillator specifications are ± 100 parts per million and ± 25 parts per million.

Table 1 tabulates the values of k times $R_{\min}$ and k times $R_{\max}$ for common values of N and L. The designer can use this table to find values of N and L that satisfy the inequalities for $R_{\min}$ and $R_{\max}$ described above for a selected value of k. Consider, for example, the design of a system containing a 5-megahertz microprocessor, an adjustable-rate realtime clock that nominally increments every microsecond, and a common crystal oscillator. Use of a common crystal oscillator implies that k should be five. Assume, further, that the worst-case

Table 1. Table of $k\ R_{min}$ and $k\ R_{max}$ versus N and L.

		Parts per million				Parts per million	
N	L	$k \times R_{min}$	$k \times R_{max}$	N	L	$k \times R_{min}$	$k \times R_{max}$
16	0	15.2590	3906.2500	19	0	1.9074	1381.0679
16	1	15.2593	2769.8182	19	1	1.9074	977.5185
16	2	15.2597	1960.8294	19	2	1.9074	691.4909
16	3	15.2607	1388.8036	19	3	1.9074	489.2396
16	4	15.2625	984.3428	19	4	1.9074	346.2273
16	5	15.2662	698.3779	19	5	1.9075	245.1037
16	6	15.2737	496.2164	19	6	1.9076	173.6005
16	7	15.2886	353.3335	19	7	1.9078	123.0429
16	8	15.3186	252.3981	19	8	1.9083	87.2972
17	0	7.6295	2762.1359	20	0	0.9537	976.5625
17	1	7.6295	1956.9584	20	1	0.9537	691.0116
17	2	7.6296	1384.9091	20	2	0.9537	488.7593
17	3	7.6299	980.4147	20	3	0.9537	345.7455
17	4	7.6303	694.4018	20	4	0.9537	244.6198
17	5	7.6313	492.1714	20	5	0.9537	173.1136
17	6	7.6331	349.1890	20	6	0.9537	122.5518
17	7	7.6369	248.1082	20	7	0.9538	86.8002
17	8	7.6443	176.6668	20	8	0.9539	61.5214
18	0	3.8147	1953.1250	21	0	0.4768	690.5340
18	1	3.8147	1382.9819	21	1	0.4768	488.5200
18	2	3.8148	978.4792	21	2	0.4768	345.5058
18	3	3.8148	692.4545	21	3	0.4768	244.3796
18	4	3.8149	490.2073	21	4	0.4768	172.8727
18	5	3.8152	347.2009	21	5	0.4768	122.3099
18	6	3.8156	246.0857	21	6	0.4769	86.5568
18	7	3.8166	174.5945	21	7	0.4769	61.2759
18	8	3.8184	124.0541	21	8	0.4769	43.4001

drift of the crystal oscillator is ± 100 parts per million and that a rate adjustment accuracy of 1 part per million is needed by the application. This means that R_{max} must be larger than 100 parts per million, and R_{min} must be less than or equal to 1 part per million. Multiplying by k, k times R_{max} must be larger than 500 parts per million, and k times R_{min} must be less than or equal to 5 parts per million. Referring to table 1, this can be accomplished by an N of 18 or more with an L of 3 or less.

While table 1 gives a solution, it is not necessarily the optimal solution. As described above, the values for R_{min} listed in table 1 were computed assuming the maximum rate adjustment counter reload value that can be represented by an N-bit rate adjustment counter with its least-significant L bits set to zero.

$$R_{\min} = \frac{1}{k\ (2^N - 2^L)}$$

In the example above, the table entry for an N of 18 and an L of 3 was selected, in part, for its k times $R_{\min}$ value of 3.8148. Let R_{low} represent the minimum rate adjustment actually required by the application. In the example above, k times R_{low} was 5 parts per million. When k times R_{low} is larger than, rather than equal to, k times $R_{\min}$, which is almost always the case, the value of x required to generate a rate adjustment of R_{low} is some number smaller than the maximum number that can be represented.

$$R_{\text{low}} \geq R_{\min}$$

$$\frac{1}{k\ x} \geq \frac{1}{k\ (2^N - 2^L)}$$

$$x \leq 2^N - 2^L$$

The value of $R_{\max}$ given that its accuracy must be the same as that for R_{low} is computed as follows

$$\Delta R = R_{\text{low}}$$

$$\frac{2^L}{k\ [\ x^2 + 2^L\ x\ ]} = R_{\text{low}}$$

$$x^2 + 2^L\ x - \frac{2^L}{k\ R_{\text{low}}} = 0$$

Solving for positive x yields

$$x = -\ 2^{L-1} + \sqrt{2^{2L-2} + \frac{2^L}{k\ R_{\text{low}}}}$$

The value of k times $R_{\max}$ is then

$$k\ R_{\max} = \frac{1}{x} = \frac{1}{-\ 2^{L-1} + \sqrt{2^{2L-2} + \frac{2^L}{k\ R_{\text{low}}}}}$$

Returning to the example above, it is clear from the table that N must be 18 or more in order to satisfy the requirement that k times $R_{\min}$ be less than or equal to 5 parts per million. The impact of L on k times $R_{\min}$ is too small to change that. But L does have a significant impact on k times $R_{\max}$. Therefore, it is advantageous to determine whether using k times R_{low}, rather than the k times $R_{\min}$ of the table, permits L to be increased. Using the equation above, an L of 4 gives

$$k\ R_{\max} = \frac{1}{-2^3 + \sqrt{2^6 + \frac{2^4}{5 \cdot 10^{-6}}}} = 561.5229 \cdot 10^{-6}$$

This new value of k times $R_{\max}$, computed from the actual k times R_{low} rather than the k times $R_{\min}$ provided by the table, is larger than the required 500 parts per million. Therefore an N of 18 or more and an L of 4 or less is a valid solution. Checking an L of 5 gives a value for k times $R_{\max}$ of 397.7926 parts per million, which being smaller than 500 parts per million, is not a valid solution. The final result, then, is an N of 18 or more and an L of 4 or less.

The width of the rate adjustment reload register and the initialized portion of the rate adjustment counter, $N - L$, can also be reduced by increasing k, the number of states in a normal cycle of the phase counter. This is attractive when the width does not neatly fit within the available hardware component partitioning. It is advantageous, for example, to replace a 17-bit register with a 16-bit register since hardware components are usually partitioned into multiples of 4 or 8 bits. The tradeoff is usually a good one because phase counter logic is relatively simple. Changing the value of k does, however, change the relationship between the oscillator frequency and the frequency at which the clock counter nominally increments.

Sometimes the rate adjustment counter and its rate adjustment reload register can be eliminated entirely by scheduling the modified phase counter cycles by using the time event services provided by the processor operating system instead. This is feasible when the application can tolerate interrupts at the minimum expected rate adjustment period.

3.4. USING MICROPROCESSOR PERIPHERAL TIMER INTEGRATED CIRCUITS

Many integrated circuit vendors supply peripheral timer integrated circuits as members of their microprocessor integrated circuit families [77]. These integrated circuits typically provide two or three counters with associated save registers to permit atomic access of their 16-bit counter values through their 8-bit processor bus port. The save registers sometimes support an automatic counter reload function for repeating counter sequences. These integrated circuits also typically provide an independent external counter clock input, external counter enable or trigger input, and discrete counter output for each respective counter. The counters are controlled through the loading of a mode register. The mode register typically selects, for each respective counter, whether the counter counts up or down, whether the count is binary or binary coded decimal, the events initiating and terminating counting, and the operation of the discrete counter output.

When the width of the clock counter and sample register can be limited to 16 bits, it would seem that these peripheral timer integrated circuits offer an opportunity to implement the clock counter with output register and the clock counter with sample register within a single integrated circuit package. Closer scrutiny, however, reveals that the modes provided by these integrated circuits are not well-suited to that approach. They do not permit the clock counter associated with the sample register to be sampled by an external strobe without also stopping the clock counter itself. Once the clock counter is stopped, there is no way to get it started again such that its value will match the value of the clock counter associated with the output register without stopping it as well. If both clock counters are stopped, then synchronization is at the mercy of software instruction execution timing.

A better approach for peripheral timer integrated circuits defines the least-significant bits of the clock value as the arithmetic sum of two of its internal counters. The external clock enable inputs for the two counters are wired such that only one counter is counting at any given time. The strobe samples the clock by toggling the selection of the counter currently counting. The strobe also generates a strobe interrupt to activate the rate adjustment algorithm as usual. Included in the processing for that interrupt is memory storage of the value of the clock that is currently not counting. The software recovers the least-significant bits of the sample value by adding the value stored for the current strobe interrupt to the value stored for the previous strobe interrupt.

[77] For example, see Z8036/8536 in Advanced Micro Devices, *MOS Microprocessors and Peripherals Data Book*, 1985; 8253 and 82C54 in Advanced Micro Devices, *Military MOS Microprocessors and Peripherals*, 1988; 8253, 8254, and 82C54 in Intel, *Peripheral Components*, 1991; MC6840 in Motorola, *8-bit Microprocessor and Peripheral Data*, 1983; DP8570A in National Semiconductor, *Real Time Clock Handbook*, 1989. The National Semiconductor DP8570A also supports time of day, day of week, and date directly in hardware. For exceptions to the typical, see Motorola, *MC68230 Parallel Interface/Timer (PI/T)*, 1983, which has one 24-bit counter, and Motorola, *MC68901 Multi-Function Peripheral*, 1984, which has four 8-bit counters. The Advanced Micro Devices Am9513A, with its five 16-bit counters and 16-bit bus, is presented at the end of this section.

Since the approach employs two internal hardware counters, and since the value in either counter is independent of the other, there are two sources of hardware counter overflow interrupts. Only one hardware counter is counting at any given time. Therefore, the *average* rate of overflow interrupt occurrence is the same as if there were only one hardware counter always counting. There only needs to be one software extension of the hardware counter value to provide the most-significant bits of the clock value. The processing in response to either internal hardware counter overflow interrupt increments the software counter. Therefore, the overflow interrupt signals from the two internal hardware counters can be gated together to form a single overflow interrupt signal.

Table 2 shows the computation of the full clock value. Each row represents the state after one cycle of the phase counter. The hardware counter values are shown in the first two columns. Initially Counter A is counting and Counter B is not. The two counter values are treated as 16-bit unsigned numbers. When added, they produce a 17-bit unsigned sum as shown in the third column. When Counter A overflows, it generates the counter overflow interrupt, which directs the software to increment the software counter. The software counter is incorporated within the translation constant shown in the fourth column. Thus, incrementing the software counter means incrementing from the least-significant bit that does not have a corresponding hardware counter bit of the same significance. The full clock value, as seen by the user, is the sum of the two counters and the translation constant as shown in the fifth column.

Table 2. Peripheral timer utilization example.

Counter A Value	Counter B Value	Sum of Counters	Translation Constant	Full Clock Value	Interrupt Received
FFF9	0004	0FFFD	770001	77FFFE	
FFFA	0004	0FFFE	770001	77FFFF	
FFFB	0004	0FFFF	770001	780000	
FFFC	0004	10000	770001	780001	
FFFD	0004	10001	770001	780002	
FFFE	0004	10002	770001	780003	
FFFF	0004	10003	770001	780004	Overflow
0000	0004	00004	→ 780001	780005	
0001	0004	00005	780001	780006	
0002	0004	00006	780001	780007	Strobe
0002	→ 0005	00007	780001	780008	
0002	0006	00008	780001	780009	

Table 2 also shows the toggle between running counters in response to the strobe. The strobe causes Counter A to stop counting and Counter B to start counting. The strobe has no effect on the normal advance of the full clock value seen in the fifth column. The strobe interrupt, among other things, stores the value of the now quiescent Counter A in memory for future use. The value stored is 0002. It retrieves from memory the value of Counter B that was stored on the previous strobe interrupt. That value is 0004. It uses the previously stored value of Counter B, rather than the current value of Counter B because Counter B is running and may advance to a different value by the time that the strobe interrupt is processed. The sample value is the sum of the value stored for Counter A, the value stored for Counter B, and the value of the translation constant at the time that the strobe occurred. In other words, the sample value is 0002 + 0004 + 780001 = 780007.

Unfortunately, the translation constant available to the strobe interrupt software is not necessarily the translation constant *at the time that the strobe occurred.* The two may differ if one or more overflow interrupts were processed between the occurrence of the strobe and the execution of the strobe interrupt software. A simple way to solve this problem is to assign a higher interrupt priority to the strobe interrupt than to the overflow interrupt so that the strobe interrupt will always be processed first. Another solution is to have an additional software counter, called the overflow adjustment counter, to keep track of the number of overflow corrections to the translation constant since the occurrence of the strobe. The overflow adjustment counter is incremented by the overflow interrupt software. It is read and then cleared by the strobe interrupt software. The strobe interrupt software uses its value to convert the current translation constant value into the value of the translation constant at the time that the strobe occurred.

A common application of the realtime clock is the measurement of time intervals between two events. The clock value for the first event is subtracted from the clock value for the second event to determine the time between events. The situation may arise where the internal hardware clock counter running when the first event time is obtained is not the same one running when the second event time is obtained. Most microprocessor peripheral timer integrated circuits do not provide the ability to read both internal hardware clock counters simultaneously. Time elapses between the reading of the first clock counter and the reading of the second clock counter. The accuracy of the time interval measurement application is therefore improved by having the software always read the running clock counter first, rather than always reading a given clock counter first. This principle is shown in figure 17.

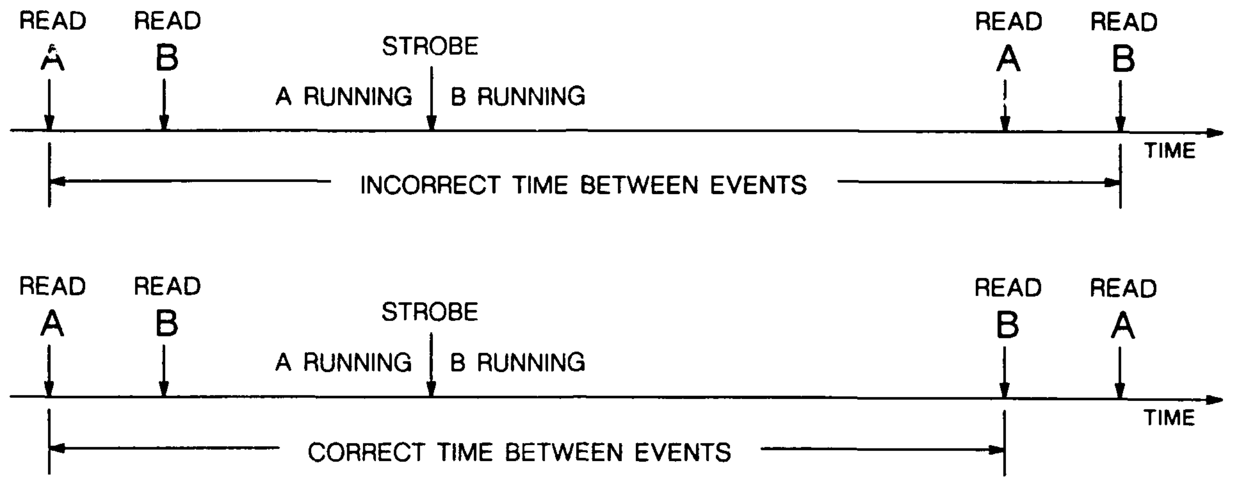


Figure 17. Peripheral timer time interval measurement.

3.5. USING THE Am9513A PERIPHERAL TIMER INTEGRATED CIRCUIT

As discussed in the previous section, the typical microprocessor peripheral timer integrated circuits have many limitations. They limit the width of the hardware clock and hardware sample values to 16 bits. They limit data bus access to 8 bits. They require external logic to implement the phase counter and the toggle between hardware counter enables. They require access of two hardware counters to compute the clock value instead of one. They require storage of previous hardware counter values in memory to recover sample values. Finally, they require software knowledge of which hardware counter is currently running so that the proper hardware counter value stored in memory can be retrieved for the recovery of the sample value and so that the measurement of the time between events can be accurately computed. On the other hand, they greatly reduce the number of integrated circuits compared to the other implementations described above.

All these limitations are overcome by using a more flexible microprocessor peripheral timer integrated circuit called the Am9513A [78]. The Am9513A provides five internal 16-bit counters. Two or more counters can be "concatenated" to form wider counters. The data bus interface is selectable for either 8-bit or 16-bit access. Commands written to a dedicated address permit the same operation to be applied to any selected set of counters simultaneously.

[78] For Am9513A data, see Advanced Micro Devices, *Military MOS Microprocessors and Peripherals*, 1988; Advanced Micro Devices, *Personal Computer Products: Processors, Coprocessors, Video, and Mass Storage*, 1989. For application and programming information, see Advanced Micro Devices, *Am9513/Am9513A System Timing Controller Technical Manual*, 1990.

A suggested allocation of the five internal counters is given below. There are, of course, many other alternatives.

Counters 1 and 2 are concatenated to form a 32-bit clock counter. This is the only clock counter. There is no need to toggle between two clock counters. Counters 1 and 2 support the least-significant and the most-significant bits, respectively. Counters 1 and 2, as opposed to the other 16-bit counters, are selected because they are the only ones supporting the alarm function. The alarm function compares the clock counter value to a preset value and generates a hardware output signal when there is a match. This signal is used as a alarm interrupt so that the software can schedule time events. Counters 1 and 2 are programmed for "count up concatenation with no gating" [79]. In this mode, the clock counter counts continuously. The mode requires no external hardware.

Counters 3 and 4 are concatenated to form a 32-bit strobe delay counter. Counters 3 and 4 support the least-significant and the most-significant bits, respectively. The strobe delay counter measures the elapsed time between reception of the hardware strobe signal from the strobe detector to access of the strobe delay counter value by the software. To recover the sample time, the software uses the "save" command to capture the values of the clock counter and the strobe delay counter simultaneously. The sample value is computed by subtracting the delay counter value from the clock counter value. Counters 3 and 4 are programmed for "count up concatenation with edge gating" [80]. In this mode, the strobe delay counter starts counting upon receiving a hardware trigger from the strobe detector and stops counting upon reaching the terminal count. The mode requires an external "D" flip-flop, inverter, and three-input NAND gate.

The hardware output signal from Counter 4 can be used as a strobe watchdog timer [81] interrupt signal. The watchdog timer time interval is selected by the initial value loaded into the strobe delay counter before the strobe signal triggers the strobe delay counter. The watchdog timer time interval value is the difference between the initial value and the terminal count. When a non-zero initial delay counter value is used, this initial value must be included in the computation that recovers the sample value since the strobe delay indicated by the delay counter is offset by the initial value of the delay counter.

Counter 5 can be assigned the phase counter function. Counter 5 is selected, as opposed to some other 16-bit counter, because it can then internally feed its phase counter output to the source input of Counter 1. Counter 5 needs to feed the source input of Counter 3 also, but this connection must be made externally. Counter 5 is programmed for the "frequency shift keying" mode, designated mode "V." This mode uses a hardware input signal to select

[79] See Advanced Micro Devices, *Am9513A/Am9513 System Timing Controller Technical Manual*, 1990, Fig. 3-1.

[80] See Advanced Micro Devices, *Am9513A/Am9513 System Timing Controller Technical Manual*, 1990, Fig. 3-3.

[81] For strobe watchdog timer, see Wilcox, NOSC TR 1400, pp. 3-5.

between two alternative periods for the counting sequence. The modified cycle signal is used to select between the normal phase counter cycle period and the modified phase counter cycle period. Since modified cycles occur infrequently, there is plenty of time for the software to load either a long or short modified phase counter cycle period value between modified cycles.

Phase counters contain only a few bits and require very little hardware to implement [82]. It is therefore usually desirable to implement the phase counter externally and use Counter 5 for some other purpose that makes better use of its 16-bit width.

Counter 5 is better employed as the rate adjustment counter. The counter supports both the rate adjustment counter and its reload register functions. It is programmed for a count down mode without gating. The Counter 5 source is either the divide-by-16 output or the divide-by-10 output of the internal Frequency Scaler [83].

Employing the notation of section 3.3., the number of states in a normal cycle of the phase counter, k, times the minimum adjustment rate, R_{min}, for the divide-by-16 case is

$$k\,R_{\text{min}} = \frac{1}{2^{16} \cdot 2^{4}} = 0.9537 \cdot 10^{-6}$$

and for the divide-by-10 case is

$$k\,R_{\text{min}} = \frac{1}{2^{16} \cdot 10} = 1.5259 \cdot 10^{-6}$$

The relationship between R_{low} and R_{max} for the divide-by-16 case is

$$k\,R_{\text{max}} = \frac{1}{-8 + \sqrt{64 + \dfrac{16}{k\,R_{\text{low}}}}}$$

and for the divide-by-10 case is

$$k\,R_{\text{max}} = \frac{1}{-5 + \sqrt{25 + \dfrac{10}{k\,R_{\text{low}}}}}$$

For a phase counter with four states per normal phase cycle, the divide-by-16 case supports adjustment rates up to 125 parts per million with a worst-case resolution of 1 part per

[82] For example, see the 2-bit phase counter in Wilcox, NOSC TR 1400 p. 17.

[83] See Advanced Micro Devices, *Am9513A/Am9513 System Timing Controller Technical Manual*, 1990, Fig. 1-15.

million, and the divide-by-10 case supports adjustment rates up to 112 parts per million with a worst-case resolution of 0.5 parts per million. Since low-cost crystal oscillators are generally specified with an accuracy of plus or minus 100 parts per million or better, the supported range is adequate for typical applications.

4.0 IEEE 802.5 STROBE DETECTOR HARDWARE

This section presents hardware implementation techniques for IEEE 802.5 local area network realtime clock strobe detectors.

4.1. SYMBOL DECODING AND ALIGNMENT IMPLEMENTATION

The strobe detector for the Texas Instruments TMS380 integrated circuit family passively monitors the connection from the media ring interface to the local area network controller. It only needs access to two wires, the receive data signal, designated RCVR, and the receive data clock, designated RCLK. For the "first generation" TMS380 integrated circuit family, RCVR is generated by the TMS38051 and RCLK is generated by the TMS38052. For the "second generation" TMS380 integrated circuit family, both RCVR and RCLK are generated by the TMS38053.

RCVR is a TTL equivalent of the differential Manchester encoded [84] signal received from the local area network media. RCVR should be sampled on the rising edge of RCLK [85]. RCLK has a one unit interval period. Its frequency is 8 megahertz for the 4-megahertz bus and 32 megahertz for the 16-megahertz bus.

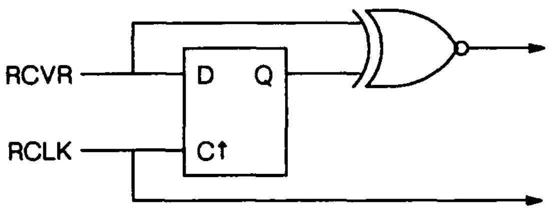
The information within a differential Manchester encoded signal is contained within the signal level *transitions*, not within the signal level itself. An inverted differential Manchester encoded signal has the same content as the original signal. The IEEE 802.5 differential Manchester encoded input can be converted into a binary sequence where two unit-interval levels represent each symbol. This conversion is accomplished with a flip-flop and an exclusive-NOR gate [86], as shown in figure 18. The table gives the first and second unit-interval values for the four types of IEEE 802.5 symbols *assuming* proper alignment of unit-interval values into symbols. Without this alignment, it is impossible to distinguish between the first and second unit intervals, and therefore impossible to uniquely decode the received input.

The alignment of unit interval values into symbols, and of symbols into octets, is determined from the unique coding of the protocol data unit starting delimiter field. The IEEE

[84] See IEEE 802.5-1989 sec 5.1, pp. 65-67.

[85] For RCVR and RCLK timing, see Texas Instruments, *TMS380 Adapter Chipset User's Guide*, Revision D, July 1986, p. 4-124, TMS38020 data sheet p. A-52. Also see Texas Instruments, *TMS380C16 & TMS38053 Data Sheet Packet: Preliminary Information*, Aug. 1990, p. 33.

[86] From the linear digital circuit point of view, the circuit "integrates" the "differential" signal. The exclusive-NOR was selected over the exclusive-OR, so that the resulting representation of the IEEE 802.5 1 and 0 symbols have active-high 1 and 0 as the first unit-interval level, respectively.



IEEE 802.5 SYMBOL	FIRST UNIT-INTERVAL LEVEL	SECOND UNIT-INTERVAL LEVEL
1	HIGH	LOW
0	LOW	LOW
J	HIGH	HIGH
K	LOW	HIGH

Figure 18. Differential Manchester decoder.

802.5 starting delimiter field is encoded as the symbol sequence "JK0JK000" [87]. If this sequence is misaligned by one unit interval, it decodes as "x11K1100," where "x" is any of the four possible symbols. The actual "x" depends on what preceded the starting delimiter symbol sequence. Since there is no "x" that produces a valid IEEE 802.5 symbol sequence, the misaligned sequence is ignored. Once a properly aligned starting delimiter is recognized, everything that follows in the same protocol data unit is assumed to have the same symbol and octet alignment.

It is not necessary to search for a match on all eight symbols of the starting delimiter. The starting delimiter is uniquely identified by the first four symbols "JK0J." The differential Manchester decoder presents these four symbols as a sequence of eight binary unit-interval values. Furthermore, it is not necessary to examine all eight unit-interval values. Consider the alternative symbol sequence "JKKJ." It differs from the sought sequence "JK0J" only in that it has a different decoded binary unit-interval value in the second unit-interval of the third symbol. Since neither "JKKJ" nor its one unit interval misalignment are valid IEEE 802.5 sequences, it is also not necessary to check the second unit-interval of the third symbol. Minimizing the number of unit-interval values checked can simplify hardware. On the other hand, seeking a match of the full starting delimiter offers a degree of fault tolerance.

It is sometimes advantageous to partition symbol alignment into a separate sequential state machine. A symbol alignment state machine can be built by noting that the only valid two-symbol sequence involving the J and K symbols is "JK" [88]. The state machine contains a shift register that captures the four most-recent sequential unit interval values. When

[87] See IEEE 802.5-1989 sec 3.2.1, p. 24.

[88] IEEE 802.5-1989, sec 5.1, p. 66, states: "To avoid an accumulating dc component, non-data symbols are normally transmitted as a pair of J and K symbols. (By its nature, a K symbol is opposite to the polarity of the preceding symbol.)" The only two examples given for non-data symbols are the starting delimiter, in sec. 3.2.1, p. 24, and the ending delimiter, in sec. 3.2.7, p. 31. Thus it is not clear what an abnormal use of the non-data symbols would be other than an error.

the sequence equivalent to "JK" appears, the symbol clock is forced to the proper symbol alignment.

An example of a schematic for such a circuit is shown in figure 19 and figure 20. It converts the raw local area network signal into a stream of symbols clocked at the symbol clock rate. The differential Manchester decoder is incorporated within the logic. The circuit is suitable for programmable array logic implementation using a PAL16R8. Its inputs are RCVR and RCLK from either the Texas Instruments TMS38051 and TMS38052, or from the TMS38053. Its outputs are SU1 and SU2, which are the first and second symbol unit interval values, respectively, and SCLK, which is the symbol clock. Figure 21 shows a timing diagram for the circuit. The symbol outputs SU1 and SU2 are stable during the rising edge of the symbol clock SCLK. The propagation delay through the circuit, measured from the rising edge of RCLK for the first unit interval value of a symbol to the rising edge of SCLK for the same symbol, is four unit intervals. This propagation delay is not dependent on previous SCLK alignment.

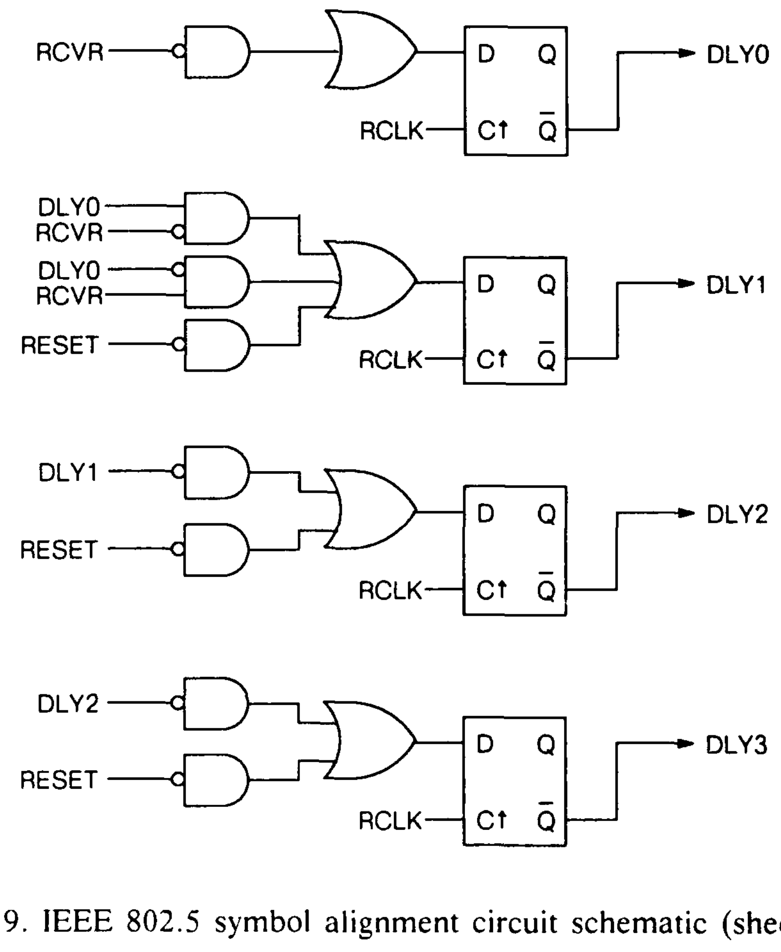


Figure 19. IEEE 802.5 symbol alignment circuit schematic (sheet 1).

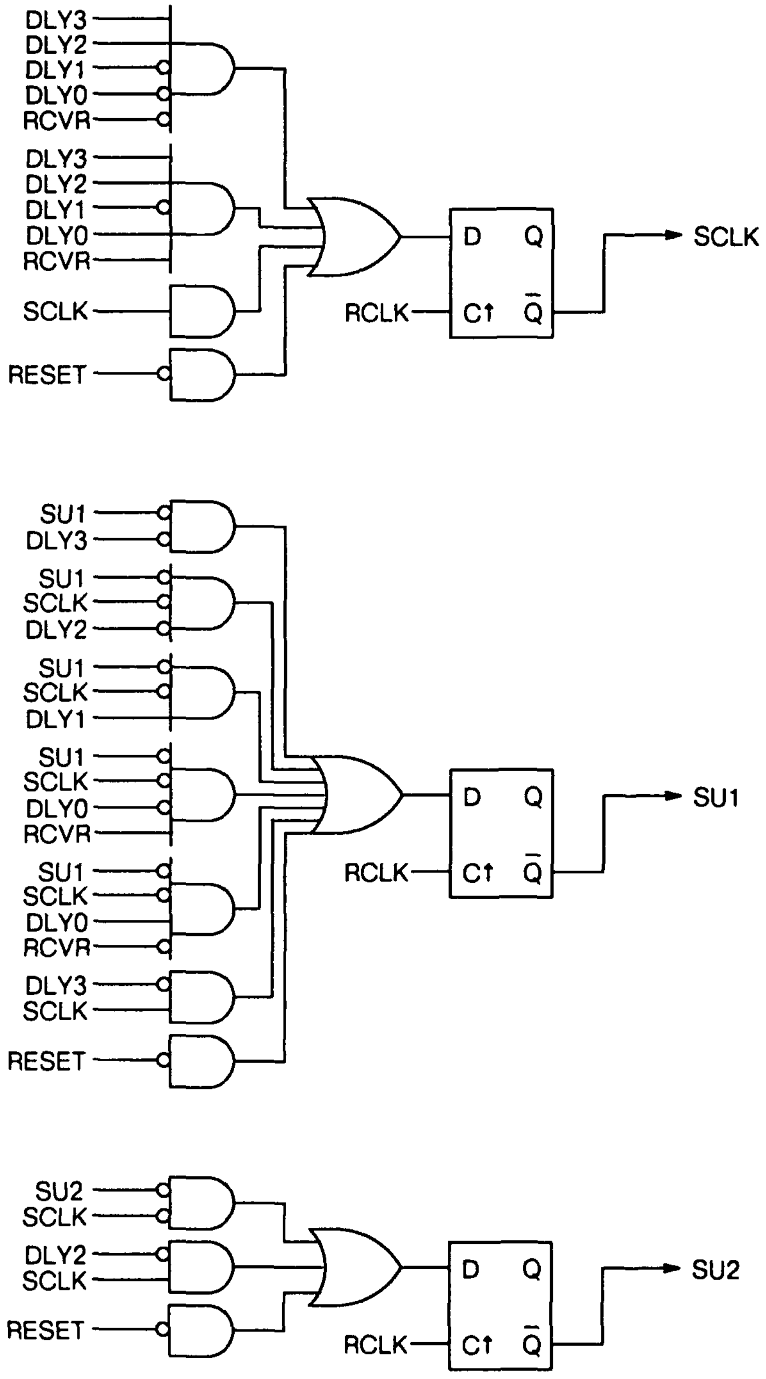


Figure 20. IEEE 802.5 symbol alignment circuit schematic (sheet 2).

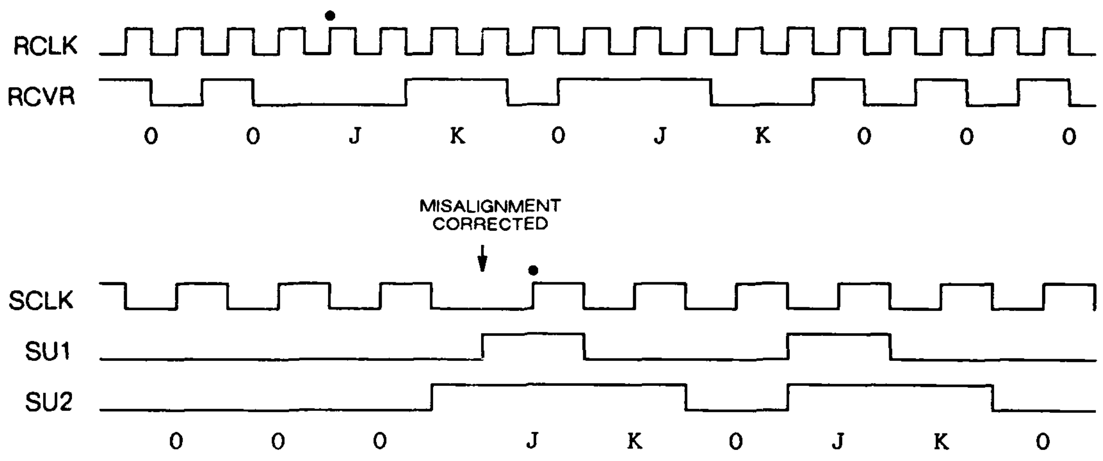


Figure 21. IEEE 802.5 symbol alignment circuit timing diagram.

The symbol alignment circuit has three advantages. First, the programmable array logic implementation of the circuit only places a single unit load on the RCVR and RCLK outputs from the TMS380 integrated circuits. The symbol clock, SCLK, which the circuit generates, clocks everything else in the strobe detector. Second, it reduces the operating frequency of the remainder of the strobe detector state machine. This is important for the 16-megahertz bus because the remainder of the strobe detector can operate at 16 megahertz rather than 32 megahertz. More low-power logic families support 16 megahertz than support 32 megahertz. Finally, it permits data bits to be recovered from the symbols 0 and 1 at the data bit rate directly from SU1 and SCLK.

4.2. PROGRAMMABLE CONTROLLER IMPLEMENTATION

The bulk of the strobe detector state machine can be constructed using a programmable controller integrated circuit such as the Am29CPL154 [89]. The Am29CPL154 does not cycle fast enough to receive a 32-megahertz RCLK directly. It can, however, receive the 16-megahertz output from the symbol alignment circuit described above.

Each clock cycle of the symbol alignment circuit potentially presents the programmable controller with any of four possible symbols. Assuming normal operation, nearly all symbols are either the symbols 1 or 0. The J and K symbols occur only during the starting and ending delimiters, and as noted previously, only starting delimiters are of interest. Since it is easier to program two-way branches rather than four-way branches in firmware, it is advantageous

[89] See 29CPL154-25 in Advanced Micro Devices, *PAL Device Data Book: Bipolar and CMOS*, 1990, pp. 3-98 – 3-132. For tutorial programming examples, see Advanced Micro Devices, *Am29PL100 Field Programmable Controllers: Handbook*, 1988.

to use external hardware to detect any unexpected J and K symbols and let the programmable controller concentrate on two-way branching in response to 1 and 0 symbols. Four-way branching is then limited to decoding the starting delimiter. As seen from the table in figure 18, the two classes of symbols are easily distinguished by the SU2 output of the symbol alignment circuit. The SU2 output is high for the J and K symbols and low for the 1 and 0 symbols.

The circuit-generating SU2, shown at the bottom of figure 20, can be replaced by one that supports detection of unexpected J and K symbols. The modified circuit, shown in figure 22, has two modes of operation. When the SU2HD input is low, the new circuit produces the same SU2 output waveform as the original circuit. When the SU2HD input is high, the behavior is the same except that once the SU2 output goes high, it remains high [90]. The programmable controller firmware sets the SU2HD input high when SU2 is low and no J or K symbols are expected. Later, at its convenience, it tests the SU2 output to determine if it is still low. If it is still low, then the SU2 output never went high, so no J or K symbols were received. If it is not still low, then at least one symbol received sometime while the SU2HD input was high was incorrectly interpreted by the programmable controller firmware as a 1 or 0 symbol when, in fact, it was a J or K symbol. Knowledge of the location of the incorrectly interpreted symbol and whether it is a J or a K symbol is lost. This is not a problem, however, since an unexpected J or K symbol, regardless of its location or value, are a signal to the firmware to abort strobe recognition.

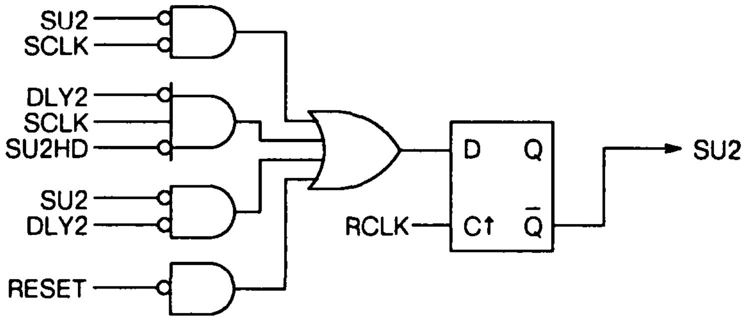


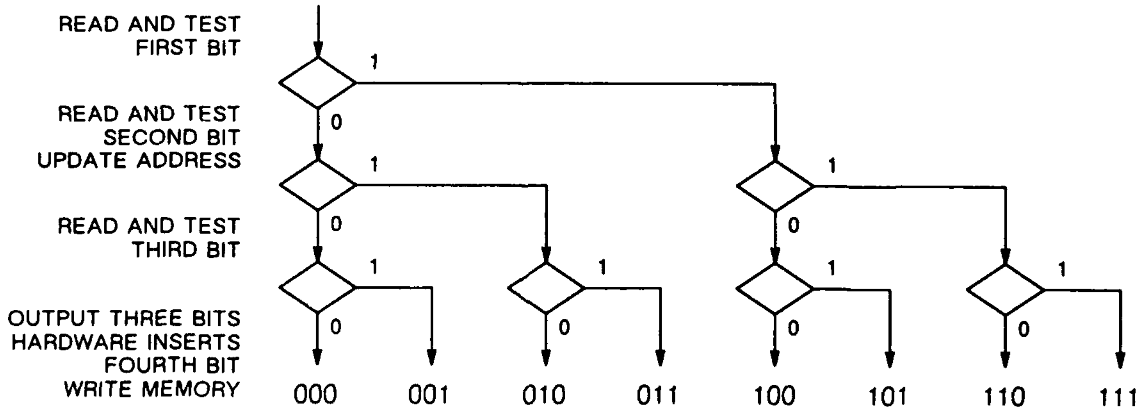
Figure 22. IEEE 802.5 symbol alignment circuit schematic (sheet 2 modification).

[90] As was the case for its predecessor, the new circuit gives an incorrect value for SU2 during symbol misalignment. This is not a problem because the programmable controller firmware does not use the SU2HD input high mode just prior to the starting delimiter where symbol alignment correction takes place.

The programmable controller interfaces to external memory to support the strobe address and strobe label registers. If there is no requirement for the strobe address to be programmable by the processor, the strobe address register memory is unnecessary. In that case, the predefined strobe address is embedded within the programmable controller firmware. Assuming that the strobe address is programmable, the strobe address register memory must be writeable by the processor and readable by the programmable controller. The strobe label register memory, on the other hand, must be readable by the processor and writeable by the programmable controller.

Atomic access is not a major concern for this memory. The strobe address register memory is only written during strobe detector initialization and read during strobe detector operation. The strobe label register memory is only written when a strobe is detected and read after the strobe is detected. There is also no conflict when the strobe address and strobe label registers reside in the same memory component since they are not addressed simultaneously by either the programmable controller or by the processor.

Storage of the strobe label requires serial-to-parallel conversion of the local area network input for storage in the strobe label register memory. One could use a hardware shift register, but that is unnecessary since the same function can be performed by the programmable controller. The programmable controller firmware, illustrated by figure 23, collects all the bits except the last bit needed for a parallel load of the memory. The *information* defined by the values of these bits, in distinction from the values themselves, is captured within the particular firmware address assumed by the program counter. The symbol alignment circuit supplies the last bit directly, as shown by figure 24. The memory load takes place in the same cycle that the last bit becomes available.



READ AND TEST FIRST BIT
READ AND TEST SECOND BIT UPDATE ADDRESS
READ AND TEST THIRD BIT
OUTPUT THREE BITS HARDWARE INSERTS FOURTH BIT WRITE MEMORY
1
0
000
001
010
011
100
101
110
111

Figure 23. Programmable controller serial-to-parallel write firmware.

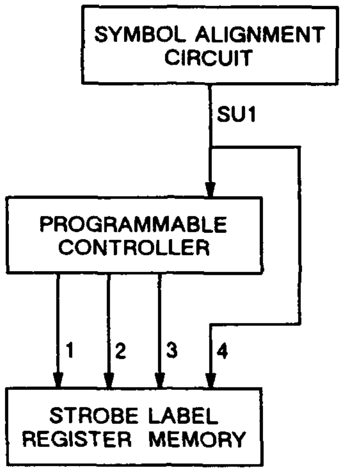


Figure 24. Programmable controller serial-to-parallel write hardware.

The number of firmware instruction locations required increases geometrically with the number of bits converted from serial to parallel. Assuming the technique described above, 15 firmware locations are required for 4-bit conversion and 255 firmware locations are required for 8-bit conversion. Obviously, 4-bit conversion is far less costly than 8-bit conversion. This motivates the use of memory components capable of being loaded in parallel 4 bits at a time.

Comparison of the strobe address with the address maintained in the strobe address register memory is more complex. As shown in figure 25, the first 2 bits received from the local area network are mapped into unique firmware memory addresses. This is the same approach as taken for the strobe label except that 2 bits, rather than 3, are mapped. The values of the first 2 bits, collected by the firmware, are sent to a 3-bit hardware identity comparator at the same time that the third bit emerges from the symbol alignment circuit. This is illustrated in figure 26. The identity comparator compares the first 3 bits received against the values of the corresponding bits in the strobe address register memory. It sends the comparison result back to the firmware at the same time that the firmware receives the fourth bit from the symbol alignment circuit. The firmware makes the final comparison using an eight-way branch. The inputs to the branch are the match output from the identity comparator, the value of the fourth bit received from the symbol alignment circuit, and the value of the corresponding fourth bit from the strobe address register memory. To provide adequate flexibility in mapping the eight-way branch needed for each 4-bit comparison into firmware instruction address space, the unused test inputs of the programmable controller

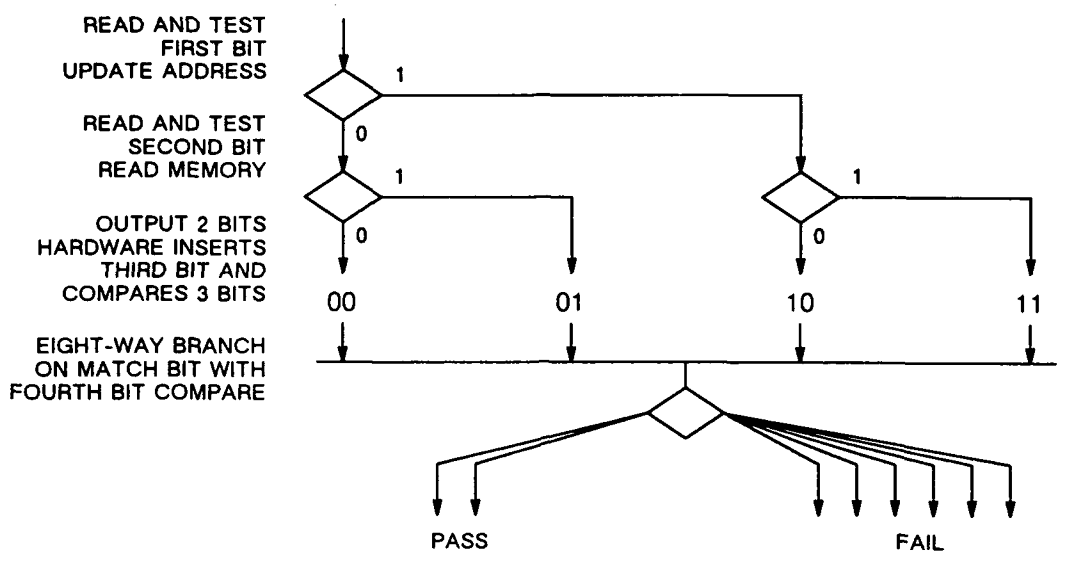

READ AND TEST FIRST BIT UPDATE ADDRESS
READ AND TEST SECOND BIT READ MEMORY
OUTPUT 2 BITS HARDWARE INSERTS THIRD BIT AND COMPARES 3 BITS
EIGHT-WAY BRANCH ON MATCH BIT WITH FOURTH BIT COMPARE
1
0
1
0
1
0
00
01
10
11
PASS
FAIL

Figure 25. Programmable controller serial-to-parallel compare firmware.

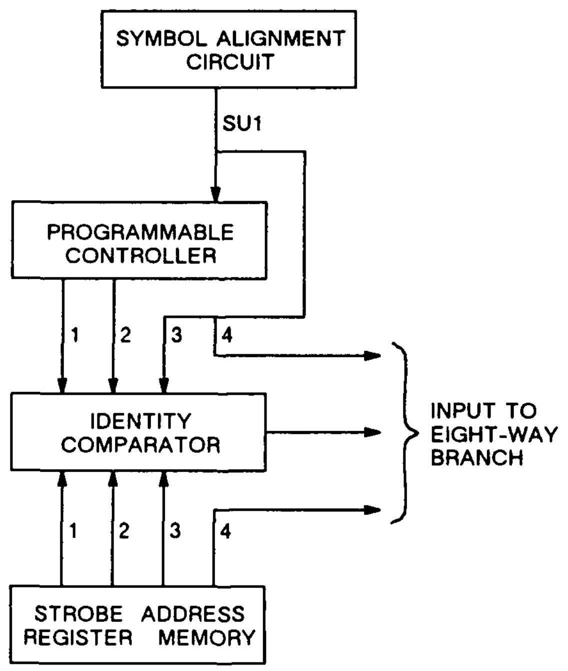


Figure 26. Programmable controller serial-to-parallel compare hardware.

are wired high. This enables the mask to selectively set the corresponding firmware address bits either high or low.

The 3-bit identity comparator can be implemented within the same PAL16R8 programmable array logic integrated circuit as used for the symbol alignment circuit. A schematic for the identity comparator is shown in figure 27. The interconnection of the symbol alignment circuit, the programmable controller, and the register memory is shown in figure 28.

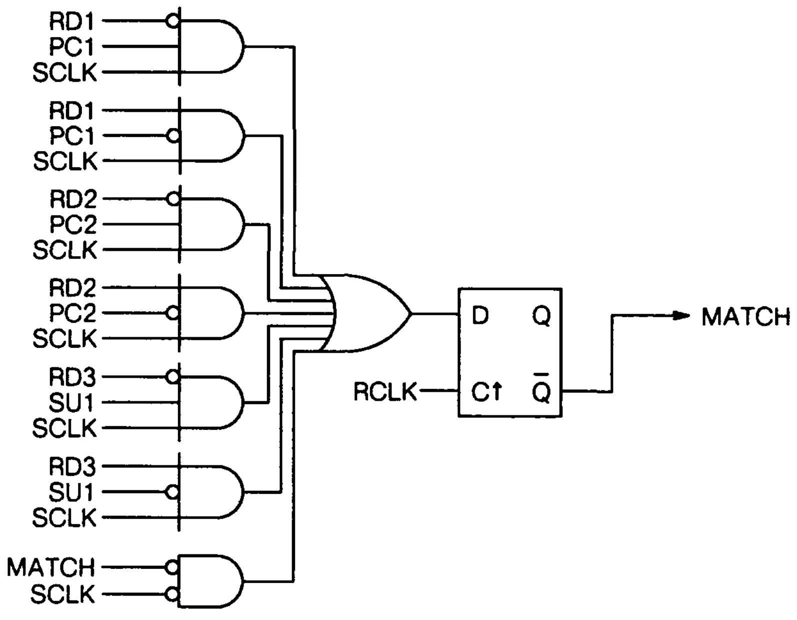


Figure 27. IEEE 802.5 symbol alignment circuit schematic (sheet 3).

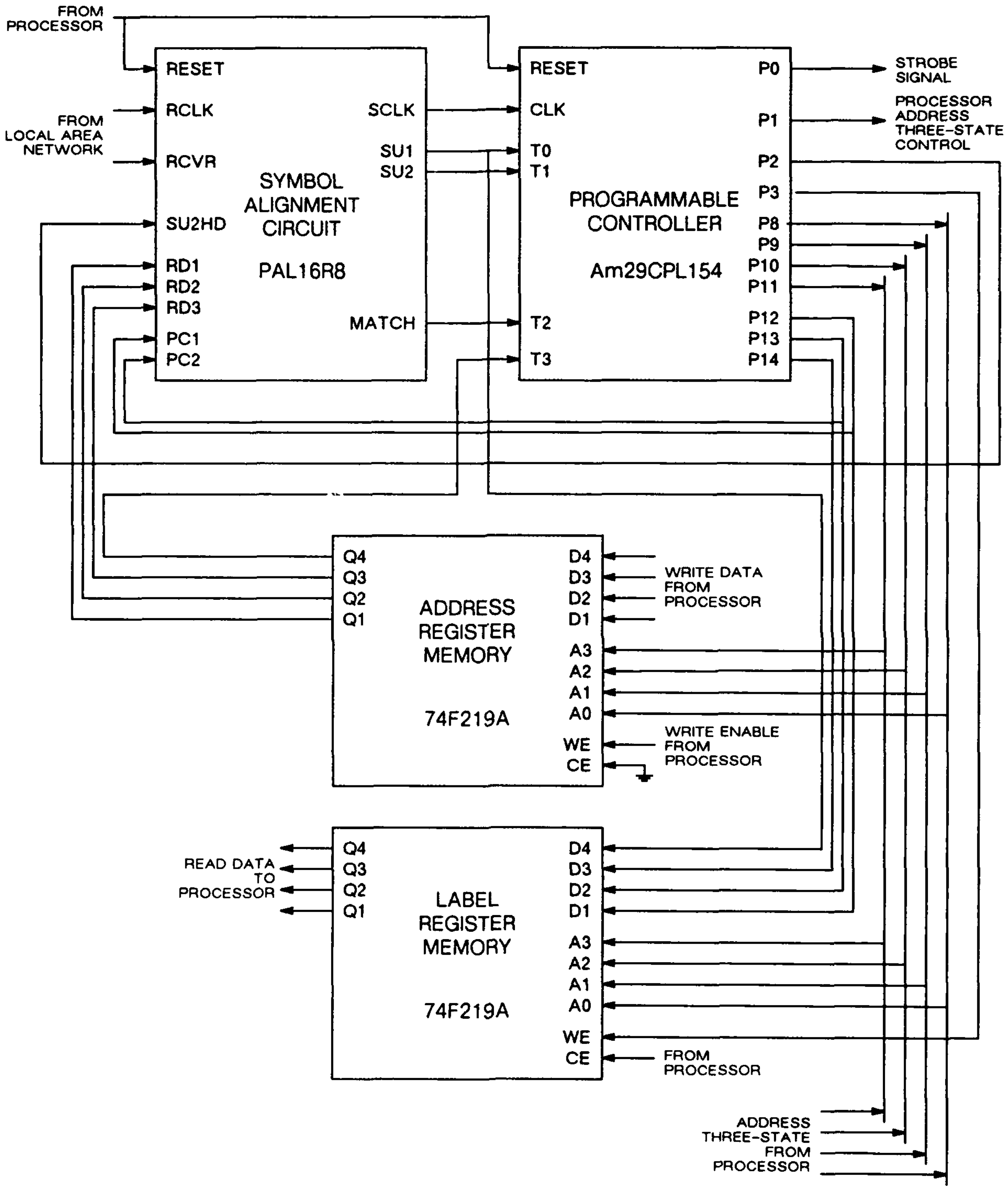


Figure 28. IEEE 802.5 strobe detector partitioning.

4.3. STROBE ADDRESS AND LABEL REGISTER IMPLEMENTATION

The selection of a strobe address register and strobe label register implementation is partly determined by the desired width of their processor interface. Typically, wide processor interfaces employ individual register integrated circuit components, while narrow processor interfaces employ small random access memory or first-in-first-out memory integrated circuit components. Small random-access memory integrated circuits, also known as register file integrated circuits, store more bits per integrated circuit component, but limit the number of bits that can be accessed simultaneously. Therefore, a wide processor interface has the advantage of faster processor access, both in hardware and software, and the disadvantage of generally requiring more integrated circuit components to implement. Since processor accesses of the strobe address register and strobe label register are infrequent, consideration should be given to a narrow processor interface to reduce hardware costs.

A second factor affecting implementation selection is whether there is a requirement to implement the strobe address register. Its implementation is required only when the strobe address needs to be programmable. Otherwise, the strobe address can be incorporated within the strobe detector firmware or hardware. The significance is more than merely one less register to implement. The strobe address register and strobe label register are accessed differently. The same strobe address register contents are read by the strobe detector state machine, at least in part, over and over again in response to each respective local area network frame processed. The strobe label register contents, on the other hand, are read by the processor only once per recognized strobe. Each new strobe replaces the strobe label register contents with new contents. Both registers can be implemented using a random-access memory, but only the strobe label register can be implemented using a first-in-first-out memory. A first-in-first-out memory cannot be used for the strobe address register because it would be impractical for the processor to constantly reload it for every frame processed. If both registers are implemented, they can share the same random-access memory component, making the first-in-first-out memory unnecessary.

Assuming that a programmable strobe address register is not required, there are two advantages to implementing the strobe label register using first-in-first-out memory rather than random-access memory. First-in-first-out memory does not require the strobe detector state machine nor the processor to generate addresses for the individual access partitions comprising the strobe label. This greatly simplifies the strobe detector state machine design since neither inline sequential firmware nor a hardware counter are needed to generate these addresses. The second advantage is that first-in-first-out memory integrated circuit components require fewer pins because there are no address pins. This allows them to fit into smaller physical packages compared to random-access memories.

First-in-first-out memory integrated circuit components, such as the 4-bit by 16-word 74F224 [91] or 74ALS232A [92], have a narrow processor interface. Wider processor interfaces can be constructed by placing them in parallel. This is illustrated by figure 29, where two 74ALS232 first-in-first-out memory integrated circuits provide a 4-bit interface to the programmable controller and an 8-bit interface to the processor. The programmable controller alternates between the two first-in-first-out memories, loading the bits that correspond to the 4 least-significant bits of a byte in one and the bits that correspond to the 4 most-significant bits of a byte into the other. For very wide processor interfaces, it is preferable to use individual shift register integrated circuits such as the 74LS299 [93] or the faster 74F299 [94] 8-bit shift registers with three-state outputs. An implementation for a 32-bit processor interface is shown in figure 30.

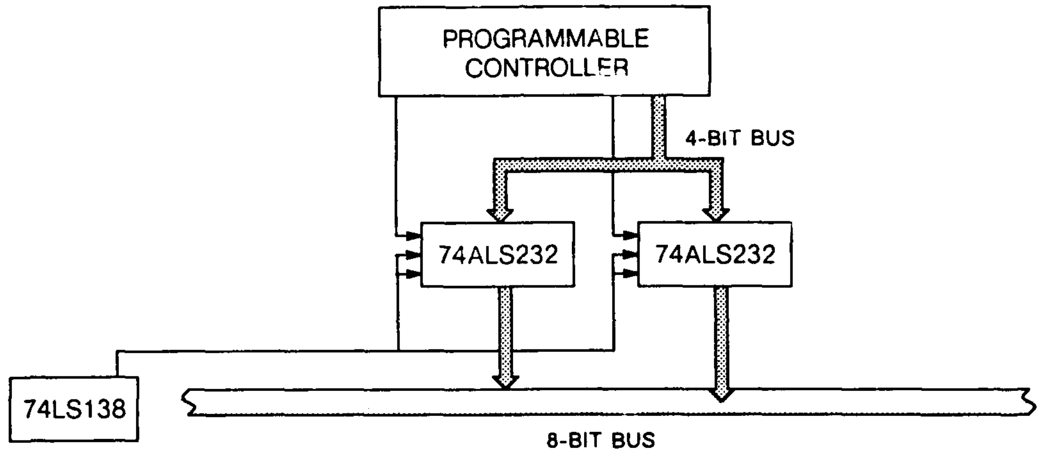


Figure 29. Strobe label first-in-first-out memory implementation.

[91] See 74F224 in Signetics, *FAST Logic Data Handbook*, 1989, pp 6-256 - 6-262.

[92] See 74ALS232A in Texas Instruments, *ALS/AS Logic Data Book*, 1986, pp. 2-255 - 2-258.

[93] See 74LS299 in Texas Instruments, *The TTL Data Book Volume 2*, 1985, 3-957 - 3-960; Motorola, *FAST and LS TTL Data*, 1989, 5-248 - 5-251. The 74LS299 easily supports the 4-megahertz IEEE 802.5 bus, but is marginal for the 16-megahertz IEEE 802.5 bus.

[94] See 74F299 in Signetics, *FAST Logic Data Handbook*, 1989, pp. 6-347 - 6-352, with Signetics, *FAST Logic Supplement*, 1990, pp. 53-55; Texas Instruments, *F Logic Data Book*, pp. 2-181 - 2-185.

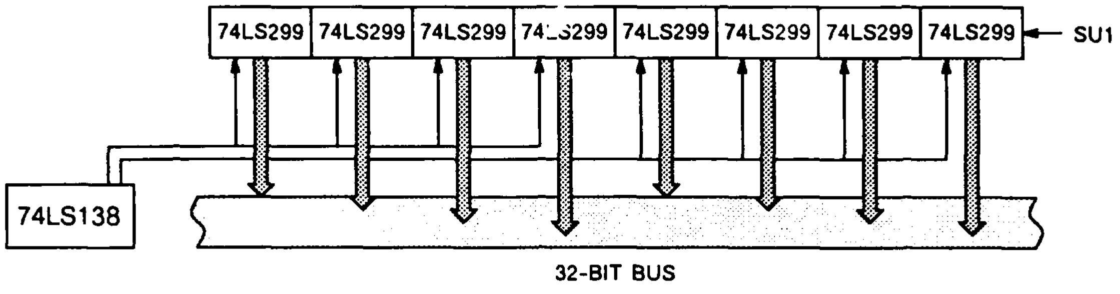


Figure 30. Strobe label shift register implementation.

The strobe address register, when implemented, requires a random access memory or a set of register integrated circuit components. Small random access memories suitable for this purpose include the 16-word by 4-bit 74F219A [95] and the dual 16-word by 4-bit dual-port 74ALS870 [96].

4.4. ALTERNATIVE STROBE GENERATION METHOD

Strobe generation is normally performed by a periodic software task executing within the processor attached to one of the nodes. The strobe itself is a local area network frame broadcast by that task to the address recognized by the strobe detectors of all the nodes participating in the clock synchronization.

The IEEE 802.5 local area network offers an alternative approach. The alternative defines the strobe as the active monitor present (AMP) [97] media access control frame. Active monitor present frames are generated by the active monitor to assure other nodes that there is a node acting as the active monitor for the ring. They are generated periodically at the request of an interval timer, called the active monitor timer (TAM) [98]. When received, an active monitor present frame resets a watchdog timer, called the standby monitor timer (TSM), at each node [99]. If the standby monitor timer expires before being reset, it is assumed that contact with the active monitor has been lost. The IEEE 802.5 specifies "default" periods for the active monitor timer and standby monitor timer of 3 seconds and 7

[95] See 74F219A in Signetics, *FAST Logic Data Handbook*, 1989, pp. 6-249 - 6-254, with Signetics, *FAST Logic Supplement*, 1990, pp. 31-36.

[96] See 74ALS870 in Texas Instruments, *The TTL Data Book Volume 3*, 1984, 2-613 - 2-617.

[97] See IEEE 802.5-1989, sec. 3.3.1.3, p. 33.

[98] See IEEE 802.5-1989, sec. 3.4.6, p. 40.

[99] See IEEE 802.5-1989, sec. 3.4.7, p. 40.

seconds, respectively. The values actually used in practice, however, are 7 seconds for the active monitor timer and 15 seconds for the standby monitor timer [100].

The advantages of defining the active monitor present frame as the strobe are (a) its ability to provide strobes without the execution of software in the attached processor, (b) its ability to identify the location of the latency buffer directly from the strobe frame source address field, and (c) its ability to guarantee strobe generation at one, and only one, node after local area network reconfiguration. The disadvantages of the approach are (a) its inability to tune the length of the sample period to optimally support the clock synchronization resolution magnitude requirements, (b) its inability to support strobes with different addresses to separate independent clock synchronizations groups, and (c) its inability to be applied to other local area network standards.

[100] These larger values are the result of an agreement among chip manufacturers participating in the National Institute for Standards and Technology (NIST) IEEE 802.5 Implementers' Workshop. See also *Survivable Adaptable Fiber Optic Embedded Network I (SAFENET I) Military Handbook*, MIL-HDBK-MCCR 0034 (Draft), 20 July 1989, sec. 7.3.1.1.1.

5.0 SUMMARY OF CONCLUSIONS

The following summary highlights some of the major conclusions of this report. Preceding sections provide more detail as well as additional conclusions.

The strobe distributed realtime clock synchronization technique, which has already been incorporated into backplane bus standards, can also be applied to token ring local area networks. The strobe is implemented as a frame protocol data unit whose destination address field is recognized by the strobe detector at all the participating nodes. The strobe frame source and information fields support the strobe label. The strobe technique does not require modification of existing local area network standards.

Strobe detection should be performed in hardware rather than in software because hardware minimizes the variation in the time between reception of the strobe and capture of the realtime clock value.

The realtime clock synchronization support hardware, when partitioned, should be partitioned at the interface between the strobe detector, which is driven by the local area network timing, and the adjustable realtime clock, which is driven by the clock oscillator timing. The strobe signal, which interfaces the two sections, can serve as an interrupt to notify the processor of the presence of a new realtime clock sample value.

Strobe detectors for serial busses, including local area networks, are implemented as hardware state machines. The state machine can be implemented using a counter. The counter starts incrementing when it sees the protocol data unit starting delimiter and resets when it encounters input failing to meet the requirements for a valid strobe. If the counter reaches its final count, it indicates detection of a valid strobe.

The worst-case strobe resolution over IEEE 802.5 is plus or minus 0.75 microseconds for the 4-megahertz bus and plus or minus 1 microsecond for the 16-megahertz bus. If the location of the active monitor at the time of strobe transmission is not known to the processor software, the worst-case strobe resolution is plus or minus 3.75 microseconds for the 4-megahertz bus and plus or minus 1.75 microseconds for the 16-megahertz bus. If nothing is known by the processor software about the current local area network ring configuration at the time of strobe transmission, worst-case strobe resolution is only slightly better than plus or minus half the maximum ring latency. The IEEE 802.5 standard specifies services allowing software to obtain the ring configuration. There is overhead associated with determining the ring configuration. If a strobe resolution of plus or minus half the maximum ring latency is good enough, then ignoring ring configuration is the preferred approach.

The worst-case strobe resolution over FDDI degrades at 0.084 microsecond per node for nodes that insert or delete single symbols and 0.164 microsecond per node for nodes that insert or delete symbol pairs.

Hidden offset method interval timers are counters decremented by the same carry signal that increments the most-significant portion of the clock counter visible to the user. Periodic phase modification method interval timers are counters decremented by the phase counter. The rate adjustment counter is an example of an interval timer. Phase accumulation method interval timers can be implemented as subtracter-accumulators similar to the clock counter adder-accumulator. This approach requires considerable hardware. By sacrificing some accuracy over long intervals, they can alternatively be implemented as counters that decrement by one on each tick cycle. The accuracy loss of a long timer interval can be compensated by dividing it into a slightly shorter anticipatory long timer interval followed by an accurate short timer interval whose length is determined using the realtime clock when the anticipatory interval expires.

The tick modification method is similar to the periodic phase modification method except that it inserts or delete ticks rather than phases to control clock rate. Contrary to appearance, it does not eliminate the need for the phase counter function. The periodic phase modification method provides better resolution than the tick modification method.

A variation of the clock value register software read atomic access approach sets the least-significant bits to zero rather than concatenating the least-significant bits read by the second access. The approach provides the correct clock value when its execution time is less than the tick period. The software read atomic access approach can be extended to any number of clock value access partitions using Lamport's algorithm.

The rate adjustment reload register needs atomic processor write access to prevent the rate adjustment counter from loading a reload value that is only partially updated by the processor. Atomic access can be implemented by disabling the rate adjustment counter during processor writes of the rate adjustment reload register.

The value-rate adjustment algorithm is a local realtime clock synchronization algorithm that uses value adjustment to correct the value error after the last adjustment and rate adjustment in an attempt to prevent accumulation of any value error at the next adjustment. It is suitable for applications where the rate of change of the realtime clock rate is small, such as for realtime clocks driven by crystal oscillators.

There are several ways to minimize the adjustable rate realtime clock hardware. The clock counter load function can be eliminated by having the software keep track of the difference between the desired clock value and the actual hardware clock value and compensating all reads of the hardware clock value by that amount before use. The width of the hardware sample register can be reduced by capturing only the least-significant bits that are needed to uniquely distinguish the sample value from the realtime clock value given the maximum delay between the time that the sample value is captured and the time that it is read. Mathematical tables and formulas can be used to determine the minimum width and scaling of the rate adjustment counter given the maximum and minimum rate adjustment and the number of phases in a normal cycle of the phase counter.

A periodic phase modification adjustable-rate realtime clock can be implemented using microprocessor peripheral timer integrated circuits. Although these integrated circuits do not implement the sample register, the same function can be obtained using two internal timers. The entire adjustable-rate realtime clock, with the exception of the phase counter and a few miscellaneous gates, can be implemented within a single Am9513A integrated circuit. The phase counter and miscellaneous gates fit easily within a small programmable array logic integrated circuit.

An IEEE 802.5 strobe detector suitable for interface to the Texas Instruments TMS380 family of integrated circuits can be constructed from four integrated circuits. The first, a PAL16R8, is used for differential Manchester decoding, symbol alignment, and implementation of a comparator to assist strobe address recognition. The second, an Am29CPL154 programmable controller, is used to recognize the sequence of symbols representing a valid strobe and to extract the strobe label from the strobe frame. The other two integrated circuits are small memories used to store the strobe address supplied by the processor and the strobe label extracted from the strobe frame. The strobe address must be stored in a random access memory. The strobe label may be stored in either a first-in-first-out memory or a random-access memory.

Additional integrated circuits may be required to interface the adjustable-rate realtime clock and strobe detector to the internal processor bus. The number varies with the design of the internal processor bus. It may be possible to implement the address decoding and control logic within the unused portion of the programmable array logic associated with the Am9513A-based implementation of the adjustable-rate realtime cloc'.:.

BIBLIOGRAPHY

This bibliography contains all the bibliography entries presented previously in NOSC TR 1400 plus new entries found or written since its publication.

CLOCKS AND CLOCK SYNCHRONIZATION

Arvind, K. 1990. "A New Probabilistic Algorithm for Clock Synchronization," Dept. of Computer and Information Science, Univ. of Massachusetts, Amherst, MA 01003.

Cole, R., and C. Forcroft. 1988, "An Experiment in Clock Synchronization," *Computer Journal,* vol. 31, no. 6, pp. 496–502.

Dolev, D., J. Y. Halpern, and R. H. Strong. 1986. "On the Possibility and Impossibility of Achieving Clock Synchronization," *Journal of Computer and System Science,* vol. 32, no. 2, pp. 230–250.

Fidge, C. 1991. "Logical Time in Distributed Computing Systems," *IEEE Computer,* vol. 24, no. 8, pp. 28–33.

Gusella, R., S. Zatti, and J. Bloom. 1986. "The Berkeley UNIX Time Synchronization Protocol," *UNIX System Manager's Manual, 4.3, Berkeley Software Distribution, Virtual VAX-11 Version*, USENIX Association, Berkeley, CA, sec. 22, pp. 1–10.

Gusella, R., and S. Zatti. 1989. "The Accuracy of the Clock Synchronization Achieved by TEMPO in Berkeley UNIX 4.3BSD," *IEEE Trans. Software,* vol. 15, no. 7, pp. 847–853.

Halpern, J. Y., B. Simons, R. Strong, and D. Dolev. 1984. "Fault-Tolerant Clock Synchroniation," *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing,* ACM, Vancouver, B.C., Canada, pp. 89–102.

Harrington, E. A. 1978. "Synchronization Techniques for Various Switching Network Topologies," *IEEE Trans. Communications,* vol. COM-26, no. 6, pp. 925–932.

IEEE P896.2. 1991. *Futurebus+ Physical Layer and Profile Specification: Draft 5.5.* IEEE, 345 E. 47th St., New York, NY 10017.

IEEE P1212. 1990. *Control Status Register Specification: Draft 4.0.*

Inter-Range Instrumentation Group. 1987. *IRIG Standard 205-87: Parallel Binary and Parallel Binary Coded Decimal Time Code Formats*, Secretariat, Range Commanders Council, White Sands Missile Range, New Mexico 88002.

Jefferson, D. R. 1985. "Virtual Time," *ACM Trans. Prog. Languages*, vol. 7, no. 3, pp. 404–425.

Kopetz, H., and W. Ochsenreiter. 1987. "Clock Synchronization in Distributed Real-Time Systems," *IEEE Trans. Computers*, vol. C–36, no. 8, pp. 933–940.

Kopetz, H., A. Damm, C. Koza, M. Mulazzani, W. Schwabl, C. Senft, and R. Zainlinger. 1989. "Distributed Fault-Tolerant Real-Time Systems: The Mars Approach," *IEEE Micro*, vol. 9, no. 1, pp. 25–40 (clock sync. pp. 31–32).

Krishna, C. M., and I. S. Bhandari. 1988. "On the Graceful Degradation of Phase-Locked Clocks," *Proceedings Real–Time Systems Symposium*, IEEE, pp. 202–211.

Lamport, L. 1978. "Time, Clocks, and the Ordering of Events in a Distributed System," *Comm. ACM*, vol. 21, no. 7, pp. 558–565.

Lamport, L. 1984. "Using Time Instead of Timeout for Fault-Tolerant Distributed Systems," *ACM Trans. Prog. Languages*, vol. 6, no. 2, pp. 254–280.

Lamport, L. 1990. "Concurrent Reading and Writing of Clocks," *ACM Trans. Computer Systems*, vol. 8, no. 4, pp. 305–310.

Lamport, L., and P. M. Melliar-Smith. 1984. "Byzantine Clock Synchronization," *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*, ACM, Vancouver, B.C., Canada, pp. 68–74.

Lamport, L., and P. M. Melliar–Smith. 1985. "Synchronizing Clocks in the Presence of Faults," *Journal ACM*, vol. 32, no. 1, pp. 52–78.

Lundelius, J., and N. Lynch. 1984. "A New Fault-Tolerant Algorithm for Clock Synchronization," *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*, ACM, Vancouver, B.C., Canada, pp. 75–88.

Mills, D. L. 1989. *Network Time Protocol (Version 2) Specification and Implementation*, Dept. Electrical Engr., Univ. Delaware, Newark, Delaware 19716, Tech. Rep. Udel–EE 89–9–2.

Molle, M. L., and L. Kleinrock. 1985. "Virtual Time CSMA: Why Two Clocks Are Better than One," *IEEE Trans. Communications*, vol. COM–33, no. 9, pp. 919–933.

National Semiconductor. 1989. "Calibration of the DP8570A Family," *Advanced Peripherals: Real Time Clock Handbook*, National Semiconductor, 2900 Semiconductor Dr., Santa Clara, CA 95052–8090, application note AN–588, pp. 2.36–2.40.

Robinson, D. C. 1989. "Using Bus Level Time Code Processors to Synchronize Multiple Signal Processing Stations," *International Telemetering Conference*, San Diego,, vol. 25, pp. 337–344; also available from Bancomm (Datum, Inc.), 6541 Via del Oro, San Jose, CA 95119.

Saltzberg, B. R., and H. M. Zydney. 1975. "Network Synchronization," *Bell Sys. Tech. Journal*, vol. 54, no. 5, pp. 879-892.

Srikanth, T. K., and S. Toueg. "Optimal Clock Synchronization," *Journal ACM*, vol. 34, no. 3, pp. 626–645.

Varghese, G., and T. Lauck. 1987. "Hashed and Hierarchical Timing Wheels: Data Structures for the Efficient Implementation of a Timing Facility," *Proceedings of the Eleventh Symposium on Operating System Principles*, pp. 25–38.

Volz, R. A., and T. N. Mudge. 1987. "Instruction Level Timing Mechanisms for Accurate Real-Time Task Scheduling," *IEEE Trans. Computers*, vol. C–36, no. 8, pp. 988–993.

Volz, R. A.; and T. N. Mudge. 1987. "Timing Issues in the Distributed Execution of Ada Programs," *IEEE Trans. Computers*, vol. C–36, no. 4, pp. 449–459.

Volz, R. A.; L. R. Sha, and D. R. Wilcox. 1991. "Maintaining Global Time in Futurebus+," *Real-Time Systems Journal*, vol. 3, no. 1.

Wilcox, D. R. 1989. "Periodic Phase Adjustment Distributed Clock Synchronization in the Hard Realtime Environment." NOSC TR 1400, Naval Ocean Systems Center, San Diego, CA 92152.

Wilcox, D. R. 1990. "Backplane Bus Distributed Realtime Clock Synchronization." NOSC TR 1400 (December), Naval Ocean Systems Center, San Diego, CA 92152.

LOCAL AREA NETWORKS

Advanced Micro Devices. 1989. *The SUPERNET Family for FDDI: 1989 Data Book*, AMD, 901 Thompson Place, Sunnyvale, CA 94088.

Advanced Micro Devices. 1989. *The SUPERNET Family for FDDI:* Technical Manual.

ANSI X3T9.5/88–139. 1990. *FDDI Media Access Control (MAC–2) (Maintenance Revision) Working Draft Proposed American National Standard.*

ANSI X3T9.5/88–148. 1990. *FDDI Physical Layer Protocol (PHY–2) (Maintenance Revision) Working Draft Proposed American National Standard.*

Clark, D. D., Jacobson, V., Romkey, J., and Salwen, H. 1989. "An Analysis of TCP Processing Overhead," *IEEE Communications*, vol. 27, no. 6, pp. 23–29.

Hutchison, J. D., C. Baldwin, and B. W. Thompson. 1991. "Development of the FDDI Physical Layer," *Digital Technical Journal*, Digital Equip. Corp., vol. 3, no. 2, pp. 19–30.

IEEE 802.5–1989. *Token Ring Access Method and Physical Layer Specification*. IEEE, 345 E. 47th St., New York, NY 10017.

MIL-HDBK-MCCR 0034 (Draft). 1989. *Military Handbook: Survivable Adaptable Fiber Optic Embedded Network I (SAFENET I)*, unapproved draft, Naval Ocean Systems Center, San Diego, CA 92152.

MIL-HDBK-0036 (Draft). 1991. *Military Handbook: Survivable Adaptable Fiber Optic Embedded Network II (SAFENET II)*, unapproved draft Jan. 15, Naval Ocean Systems Center, San Diego, CA 92152.

National Semiconductor. 1990. *DP83251/55 FDDI Physical Layer Controller: Preliminary*, National Semiconductor, 2900 Semiconductor Dr., Santa Clara, CA 95052.

National Semiconductor. 1990. *DP83261 FDDI Basic Media Access Controller: Preliminary.*

Sevok, K. C., and M. J. Johnson. 1987. "Cycle Time Properties of the FDDI Token Ring Protocol," *IEEE Trans. Software Eng.*, vol. SE-13, no. 3, pp. 376–385.

Texas Instruments. 1986. *TMS380 Adapter Chipset User's Guide*, databook SPWU001D.

Texas Instruments. 1990. *TMS380 Second-Generation Token Ring User's Guide*, databook SPWU005.

APPENDIX A: TMS380 NODE RETRANSMISSION DELAY INQUIRY

The following is the text of the response from Texas Instruments to the author [1] regarding the retransmission delay through the TMS380 components for the IEEE 802.5 local area network.

PROBLEM:

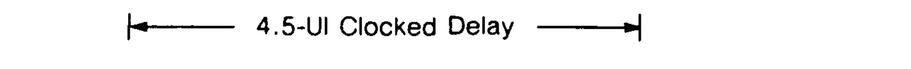
1. The data sheet in the TMS380 Adapter Chipset Users Guide (RevD) for the TMS38020 on Page A23 states "High-Speed Frame Repeat Path Minimizes Ring Latency (2 Bit Times)." Is this an accurate figure? How much jitter could one expect to accumulate per node?

2. Same question but for the "second-generation" TMS389C16 components. I have a copy of the "TMS380C16 & TMS38053 Data Sheet Packet August 1990 Preliminary Information." It gives hundreds of time parameters, but I don't seem to know where to find what I need to compute ring latency.

RESPONSE:

1) The exact clocked station delay of the first-generation chip is 4.5-UI (unit interval = 125 ns at 4 Mbit/s). In addition to this, there are propagation delays that will further delay the repeated signal. These are not characterized, but are expected to be in the 80 to 200-ns range.

Repeat path timing for first-generation (38021, 38051, 38052).



The jitter to be expected depends on the ring topography. If lobe lengths are kept to 200 m or less then, at 4 Mbits/s, the IEEE 802.5 spec currently allows only 2.0 ns of data correlated jitter and nearly 12 ns of uncorrelated jitter.

[1] Texas Instruments "TMS380 Technical Support Hotline" inquiry NAV1726A, FAX (713) 274-4027; FAX inquiry acknowledgement, 21 Aug. 1991; phone response from Steve Hubbins, 4 Sep. 1991; FAX response above, 24 Sep. 1991.

Correlated jitter will accumulate proportional to the number of nodes. Uncorrelated jitter theoretically accumulates proportional to the square root of the number of nodes.

2) The second-generation chipset has the same overall clocked station delay as the first generation although the RCLK/RCVR timing is a little different. I would also expect the propagation delay to be a little faster, perhaps 50 to 150 ns but again, we do not characterize or guarantee these. For both first- and second-generation, the propagation delay will be process and temperature dependent.

INDEX

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE November 1991	3. REPORT TYPE AND DATES COVERED Final
4. TITLE AND SUBTITLE LOCAL AREA NETWORK DISTRIBUTED REALTIME CLOCK SYNCHRONIZATION		5. FUNDING NUMBERS PE: 0600234N PROJ: RS34C76 SUBPROJ: 41-ECB1 01 WU: DN306243
6. AUTHOR(S) D. R. Wilcox		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Ocean Systems Center San Diego, CA 92152-5000		8. PERFORMING ORGANIZATION REPORT NUMBER NOSC TR 1466
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Office of Naval Technology 800 N. Quincy St. Arlington, VA 22217-5000		10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11 SUPPLEMENTARY NOTES		
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.		12b. DISTRIBUTION CODE

13 ABSTRACT *(Maximum 200 words)*

This report applies the strobe realtime clock synchronization technique to serial busses and local area networks in general, and to the IEEE 802.5 and the Fiber Distributed Data Interface token ring local area network standards in particular. Section 2 examines miscellaneous material relating to adjustable-rate realtime clocks. Section 3 presents adjustable rate realtime clock hardware implementation methods considered superior to those in an earlier report. Section 4 presents a hardware implementation of an IEEE 802.5 strobe detector.

14 SUBJECT TERMS computer architecture backplane interconnect systems realtime clock synchronization strobe detector hardware local area networks			15 NUMBER OF PAGES 88 16 PRICE CODE
17 SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18 SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19 SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20 LIMITATION OF ABSTRACT SAME AS REPORT

UNCLASSIFIED

21a. NAME OF RESPONSIBLE INDIVIDUAL D. R. Wilcox	21b. TELEPHONE *(Include Area Code)* (619) 553–5467	21c. OFFICE SYMBOL Code 412

INITIAL DISTRIBUTION

Code 0012	Patent Counsel	(1)
Code 0144	R. November	(1)
Code 402	R. A. Wasilausky	(1)
Code 41	M. L. Crowley	(1)
Code 41	A. G. Justice	(1)
Code 412	R. M. Bergman	(1)
Code 412	P. A. Casey	(10)
Code 412	J. L. Conrath	(1)
Code 412	L. J. Core	(1)
Code 412	D. K. Fisher	(1)
Code 412	P. N. Pham	(1)
Code 412	L. A. Rasmus	(1)
Code 412	D. F. Romig	(1)
Code 412	D. G. Sheriff	(1)
Code 412	V. D. Tran	(1)
Code 412	D. R. Wilcox	(30)
Code 413	L. C. Anderson	(1)
Code 413	R. E. Johnston	(1)
Code 623	B. F. Gosse	(1)
Code 6201	R. R. Eyres	(1)
Code 821	F. Duffy	(1)
Code 821	E. A. Howard	(1)
Code 952B	J. Puleo	(1)
Code 961	Archive/Stock	(6)
Code 964B	Library	(3)

Defense Technical Information Center
Alexandria, VA 22304-6145 (4)

NOSC Liaison Office
Arlington, VA 22217

Navy Acquisition, Research & Development
Information Center (NARDIC)
Alexandria, VA 22333

Navy Acquisition, Research & Development
Information Center (NARDIC)
Pasadena, CA 91106-3955

Center for Naval Analyses
Alexandria, VA 22302-0268

Office of Naval Technology
Arlington, VA 22217-5000

Space & Naval Warfare Systems Command
Washington, DC 20363-5100 (4)

Naval Air Development Center
Warminster, PA 18974-5000 (3)

INITIAL DISTRIBUTION (Cont'd)

Naval Surface Warfare Center Silver Spring, MD 20903-5000	
Naval Surface Warfare Center Dahlgren, VA 22448-5000	(3)
Fleet Combat Direction System Support Activity San Diego, CA 92147-5081	
Naval Sea Systems Command Washington, DC 20362-5101	
Naval Weapons Center China Lake, CA 93555-6001	
Naval Underwater Systems Center Newport, RI 02841-5047	(2)
Naval Weapons Support Center Crane, IN 47522-5060	
Carnegie Mellon University Pittsburgh, PA 15213-3890	(4)
University of Delaware Newark, DE 19716	
University of Virginia Charlottesville, VA 22903	
Texas A&M University College Station, TX 77843-3112	
Apple Computer Cupertino, CA 95014	
IBM T. J. Watson Research Center Hawthoren, NY 10532	
National Semiconductor Corporation Santa Clara, CA 95052-8090	
Raytheon Company Sudbury, MA 01176	
Synetics Corporation El Cajon, CA 92019	
Unisys Corporation St Paul, MN 55164-0525	
Vitro Corporation Bloomington, IN 47404	